VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

TESTOVACÍ APLIKACE PRO MIKROKONTROLER ESP32

TEST APPLICATION FOR MICROCONTROLLER ESP32

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Oliver Šintaj
AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dydowicz, Ph.D.
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání bakalářské práce

Ústav:	Ústav informatiky
Student:	**Oliver Šintaj**
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2021/22
Studijní program:	Manažerská informatika

Garant studijního programu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává bakalářskou práci s názvem:

Testovací aplikace pro mikrokontroler ESP32

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Cílem této práce je navrhnout a implementovat testovací aplikaci pro mikrokontroler ESP32 ve společnosti Elcom. s. r. o.. Úkolem této aplikace bude identifikace chybových zařízení po ukončení procesu výroby. Samotná funkčnost spočívá v testování hardwarových komponent, které jsou součástí mikrokontroleru a jimi jsou: UART a i2c sběrnice, WiFi a Ethernet modul, úložiště eMMC s následným vyhodnocením úspěšnosti testů.

Základní literární prameny:

BASL, J. a R. BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Z. Automatizované informační systémy. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

MOLNÁR, Z. Efektivnost informačních systémů. Praha: Grada Publishing, 2000. 142 s. ISBN 80-7169-410-X.

PECINOVSKÝ, R. Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky. Praha: Grada, 2009. 570 s. ISBN 978-80-247-2653-3.

SODOMKA, P. a H. KLČOVÁ. Informační systémy v podnikové praxi. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2021/22

V Brně dne 28.2.2022

L. S.

Ing. Jiří Kříž, Ph.D.	doc. Ing. Vojtěch Bartoš, Ph.D.
garant	děkan

Abstrakt

Bakalárska práca sa zaoberá analýzou a návrhom testovacej aplikácie pre mikrokontrolér ESP32 pre spoločnosť ELCOM spoločnosť s ručením obmedzeným, Prešov (ďalej uvádzaná ako Elcom). V kapitole o teoretických východísk práce sú popísane potrebné informácie pre prácu s ESP32. Nasledujúca kapitola sa zameriava na analýzu momentálnej implementácie. Posledná kapitola predstavuje návrh a formu implementácie spoločne s prínosmi nového návrhu.

Abstract

The bachelor thesis deals with the analysis and design of a test application for the ESP32 microcontroller for the company ELCOM, spoločnosť s ručením obmedzeným, Prešov (hereinafter referred to as Elcom). The chapter on the theoretical basis of the work describes the necessary information for working with ESP32. The next chapter focuses on the analysis of the current implementation. The last chapter presents the proposal and the form of implementation together with the benefits of the new proposal.

Kľúčové slová

ESP32, testovanie, I2C, eMMC, UART, WiFi, Ethernet, C++

Keywords

ESP32, testing, I2C, eMMC, UART, WiFi, Ethernet, C ++

Bibliografická citácia

ŠINTAJ, Oliver. *Testovací aplikace pro mikrokontroler ESP32*. Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/143041. Bakalářská práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Petr Dydowicz.

Čestné prehlásenie

Prehlasujem, že predložená bakalárska práca je pôvodná a spracoval/a som ju samostatne. Prehlasujem, že citácie použitých prameňov sú úplné, že som vo svojej práci neporušil/a autorské práva (v zmysle Zákona č. 121/2000 Sb., o práve autorskom a o právach súvisiacich s právom autorským).

V Brne dňa 09. mája 2022

_____

podpis študenta

Poďakovanie

Touto cestou by som sa rád poďakoval vedúcemu práce, Ing. Petr Dydowicz PhD., za odborné vedenie a rady, ktoré boli pre mňa prínosné počas písania bakalárskej práce. Tak isto by som sa rád poďakoval spoločnosti Elcom a jej zamestnancom za možnosti konzultácií pri tvorbe a implementácii danej práce.

OBSAH

ÚVOD

Zabezpečenie čo najspoľahlivejšieho a najkvalitnejšieho produktu je cieľom každej spoločnosti, ktorá svojím pôsobením prispieva do sveta vývoja a inovácií technológii. Na to, aby bolo možné dosahovať tieto ciele, je v každom procese od vývoja až po výrobu potrebné zabezpečiť čo najlepšiu možnú kontrolu, s cieľom identifikovať nedostatky v procese tvorby alebo teda už keď sa jedná o výstup týchto všetkých činností.

V spoločnosti, pre ktorú je podstata tejto bakalárskej práci navrhovaná a vyvíjaná sú tieto dôležité body známe. Preto je táto práca venovaná tvorbe aplikácie, ktorá dokáže identifikovať chybové zariadenie a tým, svojím zákazníkom ponúkať produkt čo v najvyššej kvalite a tým uspokojovať ich potreby.

Produkt, pre ktorý, je táto aplikácia vyvíjaná patri do rodiny IoT technológií. Táto technológia asi bez žiadnych pochybností, je súčasťou moderných inovatívnych procesov, ktoré nás obohacujú v každodennom živote, čo teda predstavuje zaujímavú problematiku pre túto bakalársku prácu.

VYMEDZENIE PROBLÉMU A CIEĽ PRÁCE

Cieľom tejto bakalárskej práce je navrhnúť a implementovať testovaciu aplikáciu pre mikrokontrolér ESP32 v spoločnosti Elcom. Úlohou tejto aplikácie bude identifikácia chybových zariadení po ukončení procesu výroby. Samotná funkčnosť bude spočívať v testovaní hardvérových komponentov, ktoré sú súčasťou mikrokontroléra. Tými sú: UART a i2c zbernice, WiFi a Ethernet modul a úložisko eMMC. V poslednom kroku nastane vyhodnotenie úspešnosti jednotlivých testov a tým rozpoznania zariadení, ktoré nie sú vhodné pre finálne používanie.

Implementácia sa bude realizovať pomocou programovacieho jazyka C++ na zabezpečenie vysokej efektivity a objektového spôsobu programovania.

V prvej kapitole práce sa zameriam na oboznámenie s teoretickými východiskami práce. Teoretický podklad pre vytvorenie danej aplikácie je neoddeliteľnou súčasťou tejto práce z dôvodu nutnosti pochopenia principiálnej funkcionality procesora ESP-32 a prvkov osadenými na vývojovej doske daného mikrokontroléra, komunikačnej úrovni konkrétnych zberníc, implementačných rozhraní a stavov ktoré môžu byť nadobudnuté v jednotlivých situáciách používania. Časť bude venovaná operačnému systému freeRTOS. V poslednej časti tejto kapitoly sa zamieriam na popis programovacieho jazyka C++ a princípy na ktorých je založený.

Ďalšia kapitola, ktorá nesie názov Analýza problémov a súčasnej situácie bude postavená na predstavení spoločnosti a technológií, ktoré prináša. Následne sa zameriam na analýzu súčasného stavu riešenia, ktorá bude pozostávať z identifikácie nedostatkov, hrozieb a analýza vykonávania testov, ktorá je kľúčová pre korektné pochopenie procesu realizácie. Predstavím predložené požiadavky na realizáciu a očakávaný výstup. Nasledujúci krok v rámci kapitoly pozostáva z analýzy hardvérové prvku používaného na samotné testovanie. Ukončením kapitoly bude rozbor konkurenčných riešení v oblasti testovania IoT zariadení a výstup analýzy.

Riešenie a implementácia bude obsiahnutá v poslednej kapitole. Na začiatok predstavím návrh riešenia na základe vedomostí nadobudnutých počas analýzy. Pokračovať budem popisom krokov, ktoré boli realizované počas vývoja celej aplikácie. Istá časť bude venovaná samotným testom, kde popíšem ich logiku, s čím a ako pracujú, formy

výsledkov testovania. Pre jednoduchšiu virtualizáciu krokov testu budú súčasťou algoritmy a časti zdrojových kódov, ktoré sú zodpovedné za celý proces. Tak isto budú popísane jednotlivé triedy na ktorých je postavená celá architektúra aplikácie a využívané moduly. Následne vymenujem konkrétne prínosy práce spoločne s jej ekonomickým zhodnotením.

1 TEORETICKÉ VÝCHODISKÁ PRÁCE

1.1 IoT technológie

Internet vecí alebo IoT je systém vzájomne prepojených výpočtových zariadení, mechanických a digitálnych strojov, predmetov, ktoré sú vybavené jedinečnými identifikátormi (UID) a schopnosťou prenášať dáta cez sieť. [1]

Ekosystém IoT pozostáva z inteligentných zariadení s podporou webu, ktoré využívajú vstavané systémy, ako sú procesory, senzory a komunikačný hardvér, na zhromažďovanie, odosielanie a konanie na základe údajov, ktoré získavajú zo svojich prostredí. [1]

Za posledných pár rokov sa IoT stal jednou z najdôležitejších technológií 21. storočia. [2]

Štyri kľúčové fázy internetu vecí. Zachytenie údajov, zariadenia internetu vecí prostredníctvom senzorov zachytávajú dáta zo svojho prostredia. Zdieľanie údajov, zariadenia internetu vecí pomocou dostupných sieťových pripojení sprístupňujú tieto údaje prostredníctvom verejného alebo súkromného cloudu podľa pokynov. Spracovanie údajov, v tomto bode je softvér naprogramovaný tak, aby na základe týchto údajov niečo urobil – napríklad zapol ventilátor alebo poslal varovanie. Konanie na základe údajov, analyzujú sa nahromadené údaje zo všetkých zariadení v rámci siete IoT. [3]

V bežnom živote môžeme naraziť na IoT technológie napríklad v domácej automatizácii, nositeľne monitory zdravia, biometrické bezpečnostné systémy, inteligentné auta a automatizácia procesov. [4]

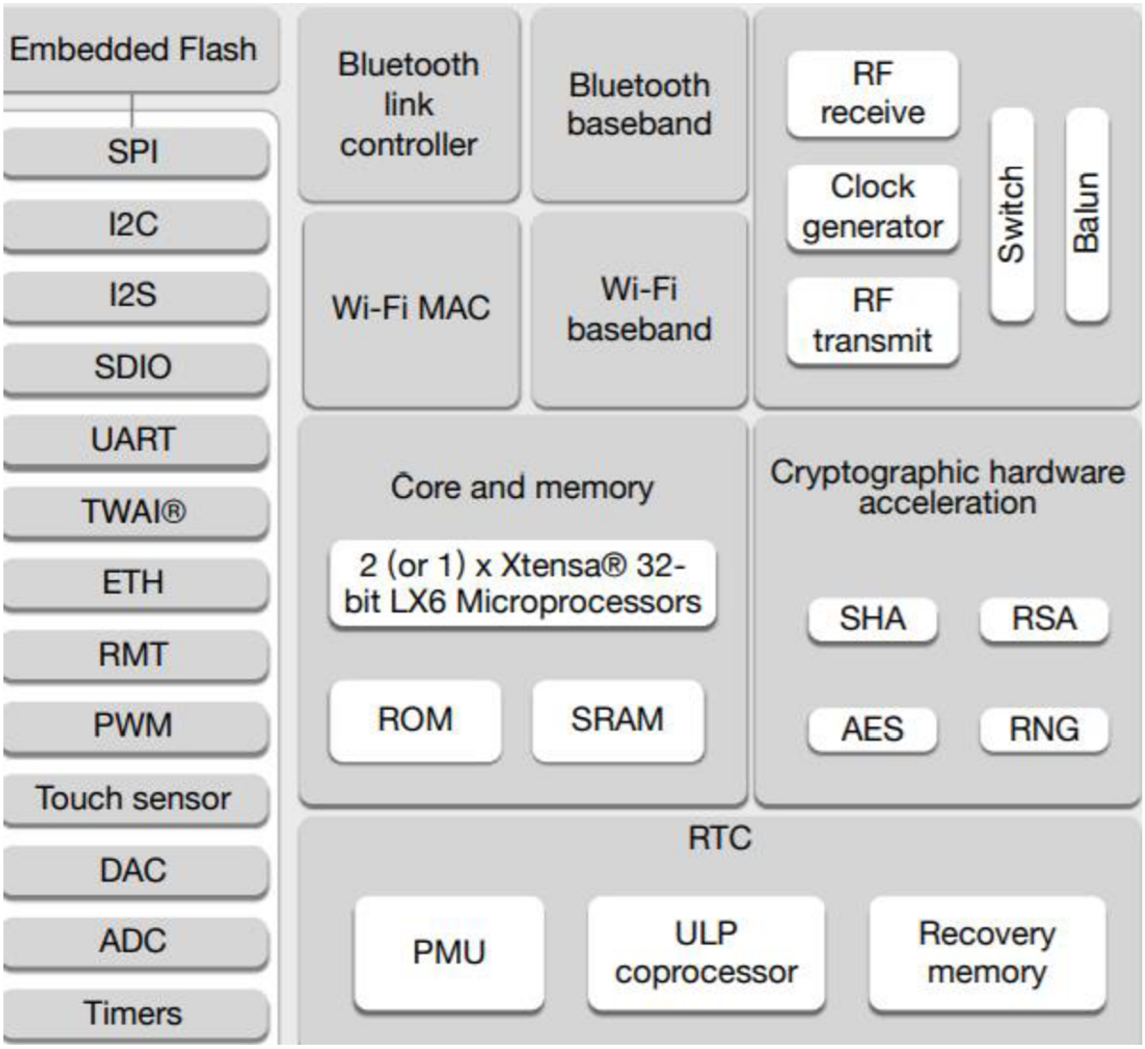
1.2 ESP32

ESP32 je jediný kombinovaný čip Wi-Fi a Bluetooth s frekvenciou 2,4 GHz navrhnutý s ultra nízkou spotrebou TSMC 40 nm technológie. Je vytvorený a vyvinutý spoločnosťou Espressif Systems. [5][8]

Séria ESP32 využíva mikroprocesor v dvojjadrových alebo v jednojadrových variáciách a obsahuje vstavané anténne prepínače, RF balun, výkonový zosilňovač, nízkošumový prijímací zosilňovač, filtre a moduly na správu napájania. [5]

ESP32 je navrhnutý pre mobilné aplikácie, nositeľnú elektroniku a aplikácie internetu vecí (IoT). [8]

1.2.1 Bloková schéma ESP32



Obrázok č. 1: Bloková schéma ESP32

(Zdroj: 8, s.12)

1.2.2 Procesor ESP32

ESP32 využíva mikroprocesor Tensilica Xtensa LX6 v dvojjadrových aj jednojadrových variáciách. [5]

Dvojjadrová verzia mikroprocesora umožňuje symetrický multiprocesing. [21]

Celá vstavaná pamäť, externá pamäť a periférie sú umiestnené na dátovej zbernici a/alebo inštrukčnej zbernici týchto CPU. [16]

Tieto dve jadrá sú pomenované Protocol CPU (PRO_CPU) a Application CPU (APP_CPU). To v podstate znamená, že PRO_CPU procesor spracováva WiFi, Bluetooth a ďalšie interné periférie ako SPI, I2C, ADC atď. APP_CPU je rezervovaný pre kód aplikácie. [14]

1.2.3 Vstavaná pamäť ESP32

Procesory majú úzko prepojenú vstavanú pamäť na konkrétne použitie. Vstavaná pamäť pozostáva zo štyroch segmentov: interná ROM (448 KB), interná SRAM (520 KB), RTC FAST pamäť (8 KB) a RTC SLOW pamäť (8 KB). Interná ROM s veľkosťou 448 KB je rozdelená na dve časti: Internal ROM 0 (384 KB) a Internal ROM 1 (64 KB). 520 KB interná SRAM je rozdelená na tri časti: interná SRAM 0 (192 kB), interná SRAM 1 (128 kB) a interná SRAM 2 (200 kB). RTC FAST Memory a RTC SLOW Memory sú implementované ako SRAM. [15] [16]

1.2.4 Integrovaná pamäť ESP32

ESP32 má tiež možnosť využiť až 4 až 8MB integrovanej pamäte RAM SPI. Táto pamäť je zakomponovaná do pamäťovej mapy a s určitými (hlavne rýchlostnými) obmedzeniami je použiteľná rovnako ako interná dátová RAM. [16]

1.3 Moduly ESP32

Espressif Systems rozdeľuje ESP32 do určitých typov modulov a nimi sú ESP32-SOLO, ESP32-WROOM, ESP32-WROVER. [7]

1.3.1 ESP32-WROOM-32

ESP32-WROOM-32 je výkonný, všeobecný Wifi – Bluetooth – Bluetooth Low Enerrgy (ďalej už iba WiFi-BT-BLE) mikrokontrolér (ďalej už iba MCU), ktorý sa zameriava na širokú škálu aplikácie, od nízkoenergetických senzorových sietí až po tie najnáročnejšie úlohy, ako je kódovanie hlasu, streamovanie hudby a dekódovanie MP3. Vložený čip je

navrhnutý tak, aby bol škálovateľný a adaptívny. K dispozícii sú dve jadrá CPU, ktoré možno samostatne ovládať a frekvencia hodín CPU je nastaviteľná od 80 MHz do 240 MHz. Čip má tiež koprocesor s nízkou spotrebou, ktorý možno použiť namiesto CPU a šetri energiu pri vykonávaní úloh, ktoré nevyžadujú veľký výpočtový výkon, ako je napríklad monitorovanie periférií. [7]

1.3.2 ESP32-WROVER-B

ESP32-WROVER-B je generický modul WiFi-BT-BLE MCU, ktoré sa zameriavajú na širokú škálu aplikácií, od nízkoenergetických senzorových sietí až po najnáročnejšie úlohy, ako napríklad kódovanie hlasu, streamovanie hudby a dekódovanie MP3. ESP32-WROVER-B sa dodáva s PCB. Je vybavený 4-16 MB integrovanou SPI flash pamäťou a dodatočnou 4-8 MB SPI pseudo statickou RAM (PSRAM). [9]

1.4 Komunikačné rozhrania ESP32

1.4.1 I2C

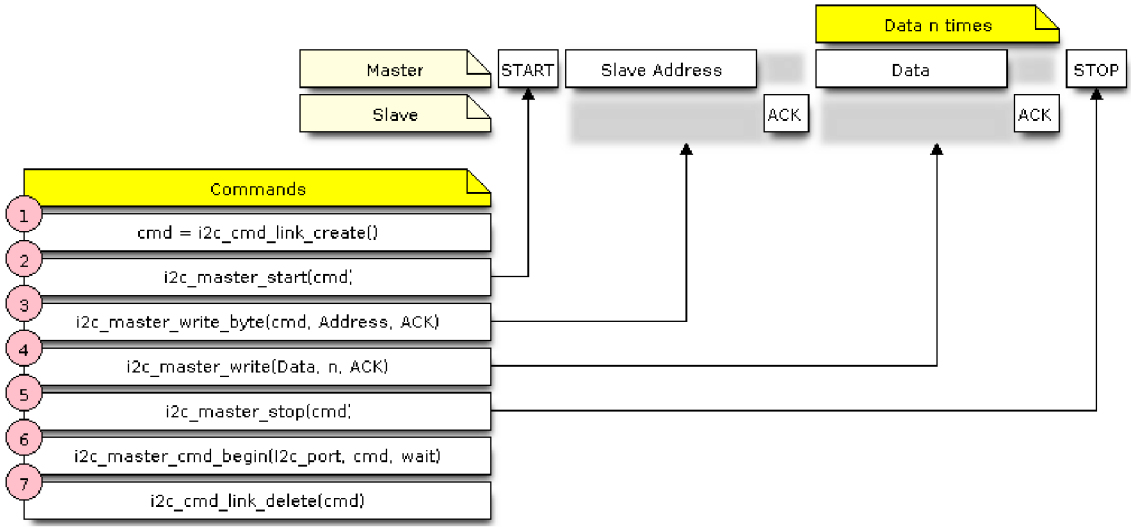
I2C (Inter-Integrated Circuit) je synchrónny, multi-master, multi-slave komunikačný protokol. Je možné ho použiť na komunikáciu s niekoľkými externými zariadeniami pripojenými na rovnakú zbernicu ako ESP32. Na doske ESP32 sú dva ovládače I2C, z ktorých každý môže byť nastavený na režim master alebo slave. [26] [10]

Komunikačný protokol I2C využíva na zdieľanie informácií dva vodiče. Nimi sú SCL a SDA. SCL je hodinová linka. Používa sa na synchronizáciu všetkých dátových prenosov cez I2C zbernicu. SDA je dátová linka. Linky SCL a SDA sú pripojené ku všetkým zariadeniam na zbernici I2C. [10] [11].

V režime master je vždy zariadenie, ktoré riadi hodinovú linku SCL. Slave sú zariadenia, ktoré reagujú na master. Slave nemôže iniciovať prenos cez I2C zbernicu, môže to urobiť iba master. Na zbernici I2C môže byť, a zvyčajne aj je, viacero podriadených zariadení, ale zvyčajne je tu iba jeden nadriadený. Je možné mať viacerých mastrov. [11]

Na organizáciu tohto procesu ovládač poskytuje kontajner, nazývaný „príkazový odkaz“, ktorý by mal byť naplnený sekvenciou príkazov a potom odovzdaný ovládaču I2C na vykonanie. [10]

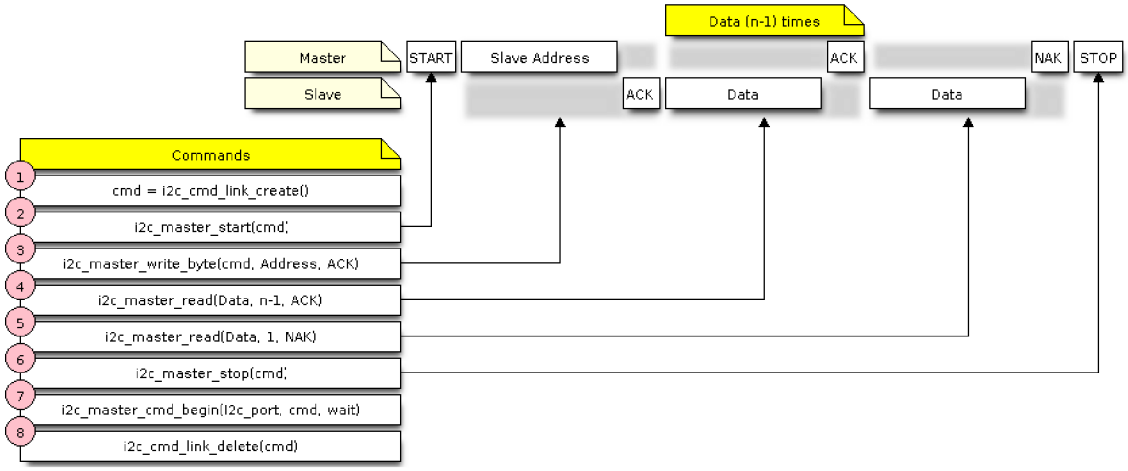
Príklad vytvorenia odkazu pre I2C master odosielajúci n bajtov podriadenému zariadeniu – Zápis mastra. [10]



Obrázok č. 2: I2C Master zápis

(Zdroj: 26)

Existuje podobná postupnosť krokov pre master na čítanie údajov z slave. [10]



Obrázok č. 3: I2C Master čítanie

(Zdroj: 26)

1.4.2 UART

UART predstavuje univerzálny asynchrónny prijímač/vysielač. Nejde o komunikačný protokol ako I2C, ale o fyzický obvod v mikrokontroléri. Hlavným účelom UART je

vysielať a prijímať sériové dáta. Poskytuje široko používanú a lacnú metódu na realizáciu plne duplexnej alebo polo duplexnej výmeny údajov medzi rôznymi zariadeniami. [12][13]

ESP32 má tri ovládače UART (UART0, UART1 a UART2), z ktorých každý má rovnakú sadu registrov na zjednodušenie programovania a väčšiu flexibilitu. Každý UART ovládač je nezávisle konfigurovateľný parametrami, ako je prenosová rýchlosť, počet dátových bitov, poradie bitov, počet stop bitov, paritný bit atď. Všetky ovládače sú kompatibilné so zariadeniami s podporou UART od rôznych výrobcov. [12]

UARTy prenášajú dáta asynchrónne, čo znamená, že neexistuje žiadny hodinový signál na synchronizáciu výstupu bitov z vysielajúceho UART so vzorkovaním bitov prijímajúcim UART. Namiesto hodinového signálu pridáva vysielací UART do prenášaného dátového paketu štart a stop bity. Tieto bity definujú začiatok a koniec dátového paketu, takže prijímajúci UART vie, kedy má začať čítať bity. Keď prijímací UART deteguje počiatočný bit, začne čítať prichádzajúce bity na špecifickej frekvencii známej ako prenosová rýchlosť. Prenosová rýchlosť je miera rýchlosti prenosu dát, vyjadrená v bitoch za sekundu (bps). Obidva UART musia pracovať približne rovnakou prenosovou rýchlosťou. Prenosová rýchlosť medzi vysielajúcimi a prijímajúcimi UART sa môže líšiť len o 3 %, kým sa časovanie bitov príliš vzdiali. Obidva UART musia byť tiež nakonfigurované na vysielanie a prijímanie rovnakej štruktúry dátových paketov. [13]

1.5 Sieťové rozhrania ESP32

ESP32 implementuje TCP/IP a plný IEEE 802.11b/g/n protokol s 2,4 GHz frekvenčným pásmom a poskytuje sadu konzistentných a flexibilných rozhraní API na podporu interného radiča Ethernet MAC (EMAC) a externých modulov SPI-Ethernet. [8] [27]

1.5.1 TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) je súbor štandardizovaných pravidiel, ktoré umožňujú počítačom komunikovať v sieti, akou je napríklad internet. [28]

TCP je jedným zo základných štandardov, ktoré definujú pravidlá internetu a je súčasťou štandardov definovaných Internet Engineering Task Force (IETF). Je to jeden z

najbežnejšie používaných protokolov v rámci digitálnej sieťovej komunikácie a zabezpečuje end-to-end doručovanie dát. [29]

IP je metóda na odosielanie údajov z jedného zariadenia do druhého cez internet. Každé zariadenie má IP adresu, ktorá ho jednoznačne identifikuje a umožňuje mu komunikovať a vymieňať si údaje s inými zariadeniami pripojenými na internet. [29]

1.5.2 IEEE 802.11x

802.11x je všeobecný pojem označujúci štandard IEEE 802.11 na definovanie komunikácie cez bezdrôtovú sieť LAN (WLAN). 802.11, bežne známy ako Wi-Fi, špecifikuje bezdrôtové rozhranie medzi bezdrôtovým klientom a základňovou stanicou alebo medzi dvoma bezdrôtovými klientmi. Tieto štandardy sa používajú na implementáciu WLAN komunikácie vo frekvenčných pásmach 2,4, 3,6 a 5 GHz. [31]

802.11b pracuje v pásme 2,4 GHz a môže poskytnúť rýchlosť až 11 Mbps s záložnou rýchlosťou na 5,5, 2 a 1 Mbps.

802.11g poskytuje maximálnu rýchlosť 54 Mbps v pásme 2,4 GHz a je spätne kompatibilný s 802.11b.

802.11n poskytuje priepustnosť až 150 Mbps pomocou priestorového multiplexovania využíva pásmo 2,4 aj 5 GHz. [31]

1.5.3 WiFi Modul

Knižnice Wi-Fi poskytujú podporu pre konfiguráciu a monitorovanie funkcií siete Wi-Fi ESP32. To zahŕňa konfiguráciu pre: Režim stanice (známy ako režim STA alebo režim klienta Wi-Fi). ESP32 sa v ňom pripája k prístupovému bodu. Režim prístupového bodu (ďalej už ako AP). Stanice sa pripájajú k ESP32. Režim koexistencie stanice/AP (ESP32 je súčasne AP a stanica pripojená k inému AP). Rôzne režimy zabezpečenia pre vyššie uvedené (WPA, WPA2, WEP atď.) Skenovanie prístupových bodov (aktívne a pasívne skenovanie) a zisťovanie sily signálu pripojenie ESP32 k AP. [30]

1.5.4 Ethernet Modul

Ethernet je asynchrónny Carrier Sense Multiple Access s protokolom/rozhraním Collision Detect (CSMA/CD). Vo všeobecnosti nie je vhodný pre aplikácie s nízkou spotrebou energie. Avšak vďaka všadeprítomnému nasadeniu, internetovému pripojeniu, vysokým prenosovým rýchlostiam a neobmedzenej rozšíriteľnosti dokáže Ethernet pokryť takmer všetky káblové komunikácie. Normálne ethernetové rámce kompatibilné s IEEE 802.3 majú dĺžku 64 až 1518 bajtov. Pozostávajú z piatich alebo šiestich rôznych polí: cieľová MAC adresa (DA), zdrojová MAC adresa (SA), pole typu/dĺžka, užitočné zaťaženie údajov, voliteľné pole výplne a kontrola cyklickej redundancie (CRC). Navyše, pri prenose na ethernetovom médiu sa k začiatku ethernetového paketu pripojí 7-bajtové pole preambuly a bajt oddeľovača SOF (Start-of-Frame). [27]

1.6 Externé úložiská

1.6.1 eMMC pamäť

eMMC (embedded Multi-Media Card) označuje balík pozostávajúci z flash pamäte a flash pamäťového radiča integrovaného na tej istej kremíkovej matrici. Riešenie eMMC pozostáva z najmenej troch komponentov – rozhrania MMC (multimediálna karta), flash pamäte a ovládača flash pamäte. [17]

Úložisko MMC pozostáva z NAND flash – tej istej technológie, akú nájdete na USB diskoch, SD kartách a SSD diskoch, no len inak zabalené. [17]

eMMC sa používa v mnohých ďalších vstavaných aplikáciách, ako sú jednodoskové počítače (SBC), robotika, zdravotnícke zariadenia, automobilový priemysel, sieťové zariadenia a zariadenia na riadenie budov, a to vďaka svojej kompaktnej veľkosti, nízkej spotrebe energie a mnohým vylepšeným funkciám. S rýchlym rastom trhu IoT zariadení si eMMC nachádza cestu k novším aplikáciám. [17]

eMMC je pripojený prostredníctvom paralelného pripojenia priamo k hlavnej doske s obvodmi akéhokoľvek zariadenia, pre ktoré ukladá údaje. Použitím integrovaného radiča v eMMC už CPU zariadenia nemusí spracovávať ukladanie dát do úložiska, pretože radič v eMMC preberá túto funkciu, takže to uvoľňuje CPU pre dôležitejšie úlohy. [17]

1.6.2 EEPROM

Elektricky vymazateľná programovateľná pamäť len na čítanie (EEPROM) je stabilný, energeticky nezávislý úložný systém, ktorý sa používa na ukladanie minimálnych množstiev údajov v počítačových a elektronických systémoch a zariadeniach, ako sú dosky plošných spojov. Tieto údaje môžu byť uložené aj bez stáleho zdroja energie ako konfiguračné alebo kalibračné tabuľky. [25]

Sériová EEPROM vo všeobecnosti funguje v troch rôznych fázach – fáza adresy, fáza dát a fáza kódu. Najznámejšie typy sériového rozhrania sú Microwire, SPI, 1-Wire, I²C a UNI/O alebo paralelné EEPROM zariadenie zvyčajne obsahuje dátovú zbernicu s ôsmimi bitmi a dostatočne širokú adresovú zbernicu na celkovú manipuláciu s pamäťou. Väčšina zariadení obsahuje piny na ochranu proti zápisu a výber čipu. [25]

1.7 Operačný systém

Operačný systém (OS) je rozhranie medzi používateľom počítača a počítačovým hardvérom. Operačný systém je softvér, ktorý vykonáva všetky základné úlohy, ako je správa súborov, správa pamäte, správa procesov, manipulácia so vstupom a výstupom a ovládanie periférnych zariadení, ako sú diskové jednotky a tlačiarne. [18]

1.7.1 Symetrický multiprocessing

Symetricky multiprocessing (ďalej ako SMP) je výpočtová architektúra, kde sú dva alebo viac rovnakých CPU (jadier) pripojené k jednej zdieľanej hlavnej pamäti a riadené jedným operačným systémom. Vo všeobecnosti systém SMP má viacero jadier bežiacich nezávisle. Každé jadro má svoj vlastný súbor registrov, prerušenia a obsluhu prerušení. Predstavuje identický pohľad na pamäť pre každé jadro. Takže kus kódu, ktorý pristupuje ku konkrétnej adrese pamäte, bude mať rovnaký účinok bez ohľadu na to, na ktorom jadre beží. Hlavné výhody systému SMP v porovnaní s jednojadrovými alebo asymetrickými multi-procesnými systémami sú, že prítomnosť viacerých CPU umožňuje viacero hardvérových vlákien, čím sa zvyšuje celková priepustnosť spracovania. Symetrická pamäť znamená, že vlákna môžu počas vykonávania prepínať jadrá. Vo všeobecnosti to môže viesť k lepšiemu využitiu CPU. Hoci systém SMP umožňuje vláknam prepínať jadrá, existujú scenáre, v ktorých vlákno musí/malo by bežať iba na konkrétnom jadre.

Vlákna v systémoch SMP preto používajú afinitu jadra, ktorá určuje, na ktorom konkrétnom jadre môže vlákno bežať. Vlákno, ktoré je pripnuté ku konkrétnemu jadru, bude môcť bežať iba na tomto jadre. Vlákno, ktoré nie je pripnuté, bude môcť počas vykonávania prepínať medzi jadrami namiesto toho, aby bolo pripnuté ku konkrétnemu jadru. [21]

1.7.2 RTOS

Operačný systém v reálnom čase, bežne známy ako RTOS, je softvérový komponent, ktorý rýchlo prepína medzi úlohami, čo vyvoláva dojem, že na jednom jadre spracovania sa súčasne spúšťa viacero programov. V skutočnosti môže jadro spracovania vykonávať iba jeden program v jednom okamihu a to, čo RTOS v skutočnosti robí, je rýchle prepínanie medzi jednotlivými programovacími vláknami (alebo úlohami), aby sa vytvoril dojem, že sa súčasne vykonáva viacero programov. [19]

Rozdiel medzi OS (operačným systémom), ako je Windows alebo Unix, a RTOS (operačný systém v reálnom čase), ktorý sa nachádza vo vstavaných systémoch, je čas odozvy na vonkajšie udalosti. Operačné systémy zvyčajne poskytujú nedeterministickú, mäkkú odozvu v reálnom čase, kde neexistujú žiadne záruky, kedy bude každá úloha dokončená, ale pokúsia sa zostať citlivé na používateľa. RTOS sa líši v tom, že zvyčajne poskytuje tvrdú odozvu v reálnom čase a poskytuje rýchlu, vysoko deterministickú reakciu na vonkajšie udalosti. Rozdiel medzi nimi je zrejmý – porovnajme úpravu napr. dokumentu na PC s obsluhou presného ovládania motora. [19]

1.7.3 FreeRTOS

FreeRTOS je trieda RTOS, ktorá je navrhnutá tak, aby bola dostatočne malá na to, aby fungovala na mikrokontroléri – hoci jej použitie nie je obmedzené na aplikácie mikrokontroléra. [20]

Pôvodný FreeRTOS (ďalej len Vanilla FreeRTOS) je malý a efektívny operačný systém v reálnom čase podporovaný na mnohých jednojadrových MCU. Avšak mnohé verzie ESP (ako napríklad ESP32) sú schopné SMP. Preto je verzia FreeRTOS použitá v ESP (ďalej len ESP-IDF FreeRTOS) upravenou verziou Vanilla FreeRTOS . Tieto úpravy umožňujú ESP-IDF FreeRTOS využívať dvojjadrové SMP schopnosti ESP. [21]

Vo verzii ESP-IDF FreeRTOS (narozdiel od Vanilla FreeRTOS) je možné určiť, na ktorom jadre bude úloha (alebo vlákno) pripnuté. Anatómia úloh je rovnaká. V ESP-IDF FreeRTOS pri odstraňovaní úlohy, ktorá je pripnutá k inému jadru, sa pamäť tejto úlohy vždy uvoľní nečinnou úlohou druhého jadra. Pri odstraňovaní úlohy, ktorá je momentálne spustená na druhom jadre, sa na druhom jadre spustí proces odovzdania a pamäť úlohy sa uvoľní jednou z nečinných úloh (v závislosti od afinity jadra úlohy) Pamäť odstránenej úlohy sa okamžite uvoľní, ak úloha momentálne beží na tomto jadre a sú tiež pripnuté k tomuto jadru alebo úloha momentálne nie je spustená a nie je pripnutá k žiadnemu jadru. [21]

Vo Vanilla FreeRTOS, keď plánovač vyberie novú úlohu na spustenie, vždy vyberie aktuálnu úlohu v stave pripravenosti s najvyššou prioritou. V ESP-IDF FreeRTOS bude každé jadro nezávisle plánovať spustenie úloh. Keď konkrétne jadro vyberie úlohu, jadro vyberie úlohu v stave pripravenosti s najvyššou prioritou, ktorú môže jadro spustiť. [21]

1.8 Jazyk C

C je vysokoúrovňový a univerzálny programovací jazyk, ktorý je ideálny na vývoj firmvéru alebo prenosných aplikácií. Pôvodne bol určený na písanie systémového softvéru, bol vyvinutý v Bell Labs Dennisom Ritchiem pre operačný systém Unix na začiatku sedemdesiatych rokov. C, zaradený medzi najpoužívanejšie jazyky, má kompilátor pre väčšinu počítačových systémov a ovplyvnil mnoho populárnych jazykov. [23]

1.9 Jazyk C++

C++, je vysokoúrovňový počítačový programovací jazyk. Začiatkom osemdesiatych rokov vyvinutý Bjarne Stroustrup z Bell Laboratories. Je založený na tradičnom jazyku C, ale s pridaným objektovo orientovaným programovaním (ďalej ako OPP) a ďalšími možnosťami. C++, spolu s Java, sa stal populárnym pre vývoj komerčných softvérových balíkov, ktoré obsahujú viacero vzájomne prepojených aplikácií. Veľké časti mnohých operačných systémov sú napísané v tomto jazyku. C++ je považovaný za jeden z najrýchlejších jazykov a má veľmi blízko k jazykom nízkej úrovne, čo umožňuje úplnú kontrolu nad alokáciou a správou pamäte. Práve táto funkcia a mnohé ďalšie možnosti z

neho robia jeden z najnáročnejších jazykov na učenie a ovládanie vo veľkom meradle. [22]

1.10 SWOT Analýza

SWOT analýza je základným nástrojom, ktorý sa používa na vyhodnotenie súčasného stavu z rôznych hľadísk, a to z hľadiska silných a slabých stránok, príležitostí a ohrození. Zároveň načrtáva možné alternatívy budúceho vývoja, možnosti na ich využitie, prípadne ich riešenie. Táto analýza je vlastne kombináciou analýzy O-T a S-W. Pričom O-T analýza predstavuje vonkajšiu (externú) analýzu, ktorá sa zameriava hlavne na vonkajšie prostredie (príležitosti a ohrozenia). S-W analýza predstavuje vnútornú (internú) analýzu, v ktorej ide o rozbor vnútorných faktorov organizácie (silných a slabých stránok). Cieľom SWOT analýzy je posúdenie vnútorných predpokladov podniku k uskutočneniu určitého podnikateľského zámeru a podrobenie rozboru i vonkajších príležitostí a obmedzení určované trhom. [33]

Cieľom SWOT analýzy je posúdenie vnútorných predpokladov podniku k uskutočneniu určitého podnikateľského zámeru a podrobenie rozboru i vonkajších príležitostí a obmedzení určované trhom. V súčasnosti neexistuje žiadna firma izolovane od okolitého sveta. Nachádza sa uprostred všetkého diania a pôsobí na ňu mnoho negatívnych a pozitívnych vplyvov. Tie ktoré prevažujú, rozhodujú o budúcnosti firmy. Záleží len na tom, ako je na rôzne vplyvy spoločnosť pripravená a ako sa s nimi dokáže vysporiadať. SWOT analýza hodnotí silné (strenghts), slabé (weaknesses) stránky spoločnosti, hrozby (threats) a príležitosti (opportunities) spojené s podnikateľským zámerom, projektom alebo stratégiou. [33]

2 ANALÝZA SÚČASNÉHO STAVU

2.1 Predstavenie spoločnosti

Spoločnosť Elcom pôsobí na slovenskom trhu už viac ako štvrťstoročie. Počas svojho dospievania vyrástla z distribútora japonskej firmy SHARP, až na poskytovateľa vlastných pokladničných riešení a dnes zaujíma na slovenskom i zahraničnom trhu jedno z popredných miest. Výber kvalitných materiálov a modernej technológie ponúkajú spoločnosti priestor pracovať nielen na vývoji a výrobe registračných pokladníc, ale aj zariadení do extrémnych podmienok, akými sú riadiace systémy v jadrových elektrárňach, elektronika pre automobilový priemysel či zdravotníctvo. [32]

2.2 Využitie ESP32 v produktoch spoločnosti

Ako som uviedol v predošlej podkapitole, jedným z produktov, ktorý je vyvíjaný a vyrábaný slovenskou spoločnosťou Elcom je registračná pokladnica. Samozrejme, tento produkt je nám všetkým jednoznačne známy, predstavuje súčasť nášho každodenného života či už v potravinách alebo v obchodoch a dokonca aj na našej fakulte, v našom fakultnom bufete.

To je práve výrobok, ktorý má v sebe zabudovaný tento mikrokontrolér a využíva vymoženosti IoT zariadení vo všeobecnosti ako je napríklad WiFi pripojenie, na zabezpečenie online formy dokladov, ktoré podliehajú zákonným predpisom jednotlivých štátov, kde sa registračná pokladnica využíva. Ďalšou podstatnou funkcionalitou je cloudová komunikácia pokladnice s Elcom cloudom, čo je jednou z služieb Elcom registračných pokladníc a množstvo ďalších funkcionalít.

Vo všeobecnosti je možné povedať to, že ESP32 v rámci registračných pokladníc teda slúži na zabezpečenie zákonných požiadaviek danej krajiny a tak isto ako rozšírenie funkcionalít pre zákazníka a jednoduchšiu prácu s registračnou pokladnicou.

Ako bolo spomínane v prvej kapitole, rada ESP32 predstavuje niekoľko špecifických modulov, v rámci registračných pokladníc sú zastúpené dva z nich. Jedna skupina pokladníc v sebe obsahuje zabudovaný typ modulu ESP32-WROOM-32 a druhá skupina ESP32-WROVER-B.

2.3 Predstavenie požiadaviek na analýzu a návrh aplikácie

Spoločnosť Elcom má svoje požiadavky na svoje produkty jasné. Je potrebné zabezpečiť 100% hardvérovú funkčnosť produktov, ktoré sú poskytované koncovým užívateľom.

Proces, ktorý zabezpečuje bezchybnosť produktov existuje a bol využívaný pri výrobe zariadení ESP32 v predošlom období a prebiehal bez problémov a poskytoval obraz o výsledku výrobného procesu tak ako bolo ním zaumienené.

Samotný mikrokontrolér prešiel istým interným vývojom a zmenami od prvotnej implementácie, kde sa vlastne pridalo veľké množstvo nových vymožeností a funkcionalít, ktoré sa odrazili na zvýšení pamäťových nárokov. Postupne sa prišlo k zisteniu, že v prípade budúcej výroby, by testovanie bolo nerealizovateľné z ohľadom na tie zmeny, ktoré sa udiali.

Ako reakciou na túto skutočnosť, aby bolo v dohľadnej dobe bolo možné využívať testovanie po ukončení výrobného procesu a aby to pomohlo identifikovať čo najskôr chybné produkty, ešte pred testovaním v rámci funkčnosti celej registračnej pokladnice. Bola vytvorená požiadavka vyhotoviť analýzu momentálnej implementácie testovacej aplikácie mikrokontroléra, jej nedostatky a problémy, ktoré spôsobujú jej nefunkčnosť, oboznámiť sa s procesom testovania počas výroby a aj zariadení, ktoré vykonávajú samotnú činnosť testovania a na základe výstupov týchto čiastočných analýz navrhnúť efektívne, rýchle a čo najjednoduchšie riešenie s maximálnym ohľadom na pamäť zariadenia. Teda postaviť túto aplikáciu na OOP spôsobe programovanie pomocou jazyka C++, ktorý svojím implementačným rozhraním umožňuje tieto podmienky splniť.

2.4 Analýza testovacej aplikácie

V tejto podkapitole sa zameriam na podrobnú analýzu všetkých častí, ktoré predstavujú súčasť testovacej aplikácie a celého procesu so všetkými náležitostiam, ktoré k tomu bezodkladne patria.

Samotnú analýzu testovacej aplikácie je možné pokladať za primárnu časť tejto práce z dôvodu toho, že je potrebné nadobudnúť obraz o spôsobe implementácie ako bola vyhotovená a následne na základe týchto znalosti, istými postupmi odhaliť problematické časti a následne ponúknuť výstup vo forme záveru tejto kapitoly pre návrh riešenia.

Pochopenie testovacieho procesu je tak isto značne podstatná časť tejto analýzy aby bolo jasné ako práca s touto aplikáciou vyzerá, aké prostriedky sú nato využívané.

2.4.1 Momentálna implementácia aplikácie

V prvom kroku, je potrebné uviesť podstatnú informáciu a to je, že táto momentálna testovacia aplikácia momentálne nie je samostatná. Jej implementácia je súčasťou firmvéru, ktorý reprezentuje reálnu činnosť fungovania mikrokontroléra spojeného s registračnou pokladnicou.

Prístup alebo teda samotné spustenie testovacej aplikácie nie je možné v produkčnom prostredí zákazníkom, v prípade oficiálneho firmvéru je táto funkcionalita znemožnená.

Na začiatok je potrebné uviesť, že samotné rozhranie testovacej aplikácie je postavené na objektovo-orientovanom programovaní, rovnako ako aj ostatné časti firmvéru, ktoré predstavujú istú podpornú funkcionalitu.

2.4.2 Jednotlivé testy

Testovacia aplikácia vo všeobecnosti v sebe zahŕňa konkrétne testy na komunikačné rozhrania I2C a UART, sieťové moduly WiFi a Ethernet a úložisko eMMC.

Každý test týchto časti mikrokontroléra je rozdelený do jednotlivých funkcii, kde sa jeho funkcionalita kontroluje istými postupmi, okrem UART komunikácie, ktorý slúži na monitorovanie výsledkov.

I2C test prebieha v procese odosielanie náhodnej sady bytov do EEPROM a následným vyčítaním týchto dát a porovnaním či sa jedná o tie isté dáta a či sú usporiadané v tom istom poradí, v akom boli vytvorené. WiFi modul sa pokúša pripojiť k AP a následne na základe sily signálu zisťujeme, či sa to podarilo dosiahnuť. Ethernet sa pripojí k portu a testuje, či bola zariadeniu priradená IP adresa. eMMC test je postavený na ukladaní fyzického súboru s konkrétnym obsahom, následne otvorenie a vyčítanie obsahu tohto súboru a porovnanie s tým čo bolo uložené a nakoniec vyhodnotenie úspešnosti jednotlivých testov.

UART je testovaný v takej forme, že všetky operácie spojené s testovaním, napríklad ich samotné spustenie a podanie nejakého výstupu o úspešnosti alebo neúspešností je

prenášané prostredníctvom UART, čiže je tým možné usúdiť, že tento komponent je v poriadku.

2.4.3 Rozdelenie testovacej aplikácie

Keďže samotný mikrokontrolér sa využíva v dvoch moduloch s odlišným zapojením pinov procesoru, tak bolo potrebné identifikovať a nakonfigurovať aj rozdiel medzi týmito modulmi v rámci testovania.

V ohľade na testovanie je tam jediný rozdiel a to taký, že moduly ESP32-WROOM-32 nevyužíva ethernet ako typ pripojenia k internetu a ESP32-WROOVER-B má v sebe túto vymoženosť zadefinovanú, ostatné hardvérové komponenty majú totožné. Z pohľadu HW zapojenia je rozdielovým parametrom konfigurácia pinov pre I2C zbernicu, pretože modul ESP32-WROOVER-B využíva pôvodné I2C piny na riadenie prístupu k integrovanej PSRAM.

2.4.4 Podporné triedy a moduly pre testovanie

Samotná testovacia aplikácie využíva z produkčného firmvéru konkrétne SW moduly a triedy na podporu vlastnej funkcionality, ktoré spôsobujú závislosť testovacej aplikácie na celkovom firmvéri. Neprítomnosť týchto tried alebo modulov, by anulovala funkčnosť testovacej aplikácie. Naopak, ich využitie vedie k potvrdeniu, že aktuálna implementácia ich dokáže korektne používať.

Pri komunikačných rozhraniach sa jedná inicializáciu I2C ovládačov, ktoré obsahujú základne nízkoúrovňové operácie ako napríklad zápis a čítanie bytov cez I2C protokol.

Sieťové komunikačné rozhrania majú samotnú triedu, ktorá obsahuje rôzne operácie, ktoré je možné vykonávať s týmito modulmi, no následne sú využívane v jednotlivých testoch, či už sa to týka konkrétne WiFi alebo Ethernetu.

Operácie, ktoré možno vykonávať s pamäťou eMMC sú tak isto definované a implementované vo vlastnej triede, ktorá sa nachádza mimo samotnej testovacej triedy, ktorá predstavuje testovaciu aplikáciu.

2.4.5 Testovacie vlákno

Samotná testovacia aplikácie sa spúšťa a vykonáva svoju činnosť v svojom vlastnom vlákne, ktoré je vyhradené na činnosti spojené s testovaním.

Toto testovacie vlákno sa aktivuje a spustí svoje konanie na základe konkrétneho vstupu z externej periférie. Môžeme si pod tým predstaviť vstupný spúšťací príkaz, zadaný z klávesnice počítača alebo formou podporného skriptu v prípade, že je UART pripojený k počítačovej jednotke.

2.4.6 Trieda testovacej aplikácie a jej metódy

Trieda testovacej aplikácie obsahuje viacero častí, ktoré sú zodpovedné za samostatnú činnosť aplikácie formou svojich metód a obsahujú aj inštancie alebo referencie jednotlivých podporných tried, premenné a inicializačné funkcie jednotlivých ovládačov.

Inštanciami alebo referenciami podporných tried, sú myslené jednotlivé triedy, ktoré obsahujú istú funkcionalitu a formou metód sa využívajú už konkrétne v testovacích procesoch. O aké triedy sa jedná bolo popísane v predošlej podkapitole, kde som uviedol o aké konkrétne sa jedná.

Z pohľadu metód ide o jednotlivé testy, ktoré som definoval v predošlej podkapitole. Každý jeden test, okrem spomínaného UARTu, je implementovaný v samostatnej metóde. Okrem týchto metód sa tam nachádza inicializačná metóda, ktorá ma za úlohu inicializovať ovládače UARTu, pre konkrétny typ komunikácie, ktorý bude prebiehať počas testovania. Všetky tieto metódy, okrem inicializačnej, sa spúšťajú v osobitnej metóde, ktorá sa spúšťa v testovacom vlákne po jeho aktivácii.

2.5 Proces testovania

Proces testovania prebieha po ukončení výrobného procesu, ktorý sa uskutočňuje vo výrobnej hale spoločnosti Elcom pomocou testovacích zariadení, ktoré umožňujú otestovať väčšie množstvo mikrokontrolérov naraz, teda svojou paralelizáciou výrazne urýchľuje proces testovania.

Samotný proces testovania a vyhodnotenia korektnosti je automatizovaný ale vyžaduje si prítomnosť jedného pracovníka na obsluhu tohto zariadenia a to v takej forme, že je

potrebné mikrokontroléry osadiť do konkrétneho prípravku, následne spustiť testovaciu aplikáciu a na základe výstupov, ktoré testovacia aplikácia poskytne vyhodnotiť funkčnosť jednotlivých osadených modulov. Moduly vykazujúce chyby sú z ďalšej produkcie automaticky vyčlenené.

2.5.1 Spustenie testovania

Zahájenie testovania vykonáva pracovník výrobnej obsluhy, ktorý spustí testovanie tak, že cez príkazový riadok spustí skript, ktorý ma v sebe potrebné konfigurácie a aktivuje testovacie vlákno na základe vstupného príkazu.

Druhý spôsob spustenia je taký, že sa UART osadený na ESP32 pripojí do počítačovej jednotky a príkazy potrebné na spustenie sa zadajú priamo pomocou klávesnice počítača.

Jedná sa o príkaz s názvom TEST, ktorý spúšťa celý proces testovania, čiže spúšťa jednotlivé testy v rade za sebou.

Je možné spustiť jednotlivé testy aj samostatne a to tak, že sa zadá konkrétny názov testu. Príkazy na spustenie samostatných testov majú názov podľa toho aký konkrétny hardvérový komponent testujú, čiže napríklad pre test eMMC je potrebné zadať EMMC a vykoná sa test na úložisko a poskytne aj výsledok.

Začiatok a ukončenie testovania je sprevádzané výpismi do príkazového riadku, aby zamestnanec bol oboznámený s procesom testovania.

2.5.2 Vyhodnotenie testovania

Samotné vyhodnotenie testovania je zobrazované v príkazovom riadku, v tom istom mieste kde bolo spustené samotné testovania na základe príkazu alebo príkazov.

Výsledky nadobúdajú dve hodnoty, PASS alebo FAILED. PASS, ktorý v preklade znamená prešiel, predstavuje úspešnosť testu a FAILED (zlyhal) hovorí o opaku, čiže vypovedá o nefunkčnosti komponentu.

Každý výsledok jedného z testov konkrétneho komponentu, či už napríklad WiFi alebo I2C je zobrazený samostatne s jednou z hodnôt výsledku, aby bolo možné identifikovať, čo konkrétne zlyhalo.

Pre jednoduchšie vnímanie, sú jednotlivé výpisy farebne upravované, PASS má zelenú farbu a FAILED červenú.

Ako som spomínal v predošlej podkapitole o testovaní samotného UARTU, ktorý je v princípe testovaný počas behu celej aplikácie tak vyhodnotenie jeho funkčnosti je taký, že testy sa podarí zrealizovať alebo nie a následne môže byť usúdené, že UART predstavuje problém.

2.6 Zariadenia využívané počas testovania

Na testovanie ESP32 spoločnosť Elcom využíva interne vyvinuté testovacie prostriedky pre jednotlivé moduly samotného mikrokontroléra.

Testovací prípravok, ktorý sa využíva po ukončení procesu výroby ma v sebe zabudovaných viacero funkcionalít, ktoré umožňujú vykonávať testovanie bez potreby prítomnosti registračnej pokladnice a je možné ho využiť hneď po tom ako sa dostane mikokontrolér do finálnej fázy svojej výroby.

Tento prípravok predstavuje zdroj napätia pre ESP32 a má v sebe zabudované EEPROM, ktoré sú potrebné pre testovanie I2C komunikácie.

Pri module ESP32-WROOVER-B je z hľadiska testovania jediný rozdiel, a to ten, že má v sebe zabudovaný aj ethernet. Z toho dôvodu, spoločnosť Elcom disponuje dvoma typmi testovacích zariadení, kde jeden má zabudovaný aj ethernet.

2.7 Identifikácia nefunkčnosti, problémov a nedostatkov testovacej aplikácie

Hlavnou a veľkou nevýhodou problematiky testovania ESP32 v spoločnosti Elcom je samotná úroveň implementácie. Konkrétne sa jedna o to, že táto aplikácie je súčasťou produkčného firmvéru. V praxi, počas používania samotného ESP32 v registračnej pokladnice sa v kódovej pamäti nachádza niečo, čo nie je v produkčnom cykle využívane a zaberá zbytočné miesto, a teda navyšuje pamäťové nároky samotného mikrokontroléra.

Počas analýzy a testovania samotnej funkcionality aplikácie sa mi podarilo odhaliť miesta vzniku chýb, ktoré sa v praxi odrazili nefunkčnosťou testovacieho procesu a tak isto dôvody, ktoré k tomu viedli.

Chybovosť sa ukázala v teste, ktorý kontroluje funkčnosť I2C komunikácie a prejavilo sa to na úrovni operačného systému tak, že nastával deadlock a proces testovania sa natvrdo zastavil a všetky procesy prestali odpovedať.

Ďalšia chyba sa prejavila pri spustení testu WiFi, kde spustenie odpovedalo neúspechom a následne po druhom pokuse, už bol test bol vyhodnotený kladne. Tento problém sa identifikoval v samotnej implantácií testu.

Nedostatkom tejto aplikácie je napríklad to, že po ukončení testov, pomocné súbory, ktoré boli vytvorené počas procesu testovania nie sú odstránené a teda sú zbytočne stále uložené aj keď ich úžitok a zmysel sa po testovaní anuluje.

Testovací prípravok spoločnosti Elcom má v sebe zabudované isté zariadenia, ktoré podporujú testovanie a predstavujú základný pilier testovania, no táto aplikácia v súčasnej verzii, nemá v sebe implementované žiadne možnosti diagnostiky toho prípravku, teda či správne funguje. To vnímam ako nevýhodu, ktorá môže mylne poskytnúť informácie pre testovací proces a spôsobiť zbytočné straty.

2.8 Konkurenčné riešenia

Pred začiatkom návrhu vlastného riešenia problematiky testovania hardvérových komponentov MCU ESP32 som venoval istý čas analýze možných riešení v rámci spoločností, ktoré sa venujú vývoju a práce s týmito zariadeniami. Musím skonštatovať, že takéto explicitné riešenie s konkrétnymi požiadavkami nie je. Je možné predpokladať, že takéto spôsoby kontroly vlastného produktu si každá firma interne vyvíja a je to súčasťou ich know-how, čiže nie je možné sa dopracovať k ich riešeniam.

2.9 SWOT analýza

V tejto kapitole, využijem SWOT analýzu na vyhodnotenie momentálnej implementácie testovacej aplikácie.

Silné stránky – možnosť otestovať niektoré z hardvérových komponentov, potrebné moduly na spustenie testovania sú prítomné v rámci produkčného firmvéru

Slabé stránky – nefunkčnosť istých testov, implementácia aplikácie je súčasťou produkčného firmvéru, kde počas využívania v produkčnom prostredí je zbytočná, vyššia

náročnosť na kódovú pamäť, prítomnosť modulov, ktoré nie sú súčasťou testovacieho procesu, nemožnosť otestovania testovacieho zariadenia

Príležitosti – možnosť rozšírenia funkcionalít aplikácie z dôvodu prítomností všetkých potrebných modulov

Hrozby – pravidelné úpravy produkčného firmvéru môžu viesť k celkovej nefunkčnosti aplikácie, nekorektný proces testovania

2.10 Záver analýzy

Táto analýza bola veľmi prínosná a podstatná na to, aby som nadobudol potrebné informácie a vedomosti, ktoré využijem pri budúcich krokoch, ktoré budú spojené so samotným návrhom riešenia a následnou implementáciou môjho návrhu.

Okrem vedomostí, týkajúcich sa testovacej aplikácie, bolo tak isto potrebné nadobudnúť ucelenejší obraz o procese testovania, čo to znamená a ako prebieha tento testovací proces a teda v budúcich krokoch zabezpečiť, aby môj návrh neviedol k následným komplikovaným zmenám, ktoré by mohli tento proces narušiť tým, že by bolo potrebné preškolenie obslužného personálu.

Analýza mi poskytla možnosť pochopiť konkrétne testy jednotlivých hardvérových komponentov a dala možnosť inšpirácie, pri budúcom návrhu vlastných testovacích procesov. V rámci týchto analýz, konkrétne testovacích metód, sa mi podarilo odhaliť chyby a problémy týchto implementácii, ktoré počas návrhu riešenia budem eliminovať.

Okrem testov samotných, bolo potrebné si skompletizovať aj podporné triedy či moduly, ktoré sú podstatné pre celkové fungovanie a vyvodiť záver, na ktorých konkrétne je táto implementácia závislá.

Postupné kroky počas tejto analýzy ma nepriviedli len k záverom, čo sa týka chýb ale tak isto mi dali nadhľad na implementáciu v širšom ponímaní a schopnosť vyhodnotiť niektoré časti ako nedostatky.

Všetky potrebné vedomosti, ktoré sú podstatné či už na prvotný vývoj softvérového produktu alebo na optimalizačné kroky, táto analýza poskytuje.

3 VLASTNÉ NÁVRHY RIEŠENIA

3.1 Návrh aplikácie na základe výstupu analýz

Po vykonaní všetkých analýz, ktoré som opisoval v predošlej kapitole, mojím výsledným návrhom pre riešenie tejto problematiky je samostatná testovacia aplikácia. To znamená na rozdiel od prvotnej implementácie, táto aplikácia bude predstavovať samostatný firmvér. Nadväznosti, ktoré existovali v predošlej implementácii , budú prevedené tak, že sa bude jednať o podporné funkčné moduly. To znamená, že oficiálny firmvér pre zariadenie ESP-32 sa stane pod-projektom tejto testovacej aplikácie a bude využívať iba moduly, ktoré sú potrebné na jeho činnosť. Takáto implementácia, využitia finálneho firmvéru formou pod-projektu predstavuje rýchlejšie riešenie, pretože nebude potrebné nanovo implementovať potrebné moduly.

Jedná sa o moduly, pracujúce so sieťovými prvkami, zbernicami, úložiskom a moduly zabezpečujúce prácu s vláknami. V rámci týchto modulov, bude potrebné zahrnúť aj moduly, ktoré priamo nesúvisia s testovaním ale sú potrebné pre fungovanie spomínaných modulov.

Spustenie samotnej aplikácie bude realizované z takzvanej hlavnej funkcie, ktorá bude zodpovedať základným štandardom ESP32 pre spustenie firmvéru na MCU. Za spustenie bude zodpovedný samotný testovací prípravok, ktorý slúži aj ako zdroj napätia pre ESP32.

Realizácia konkrétnych činností tejto aplikácia, ako napríklad spustenie testovania, bude realizované cez komunikačné rozhrania UART pomocou príkazov zadávaných z externých periférií.

Aplikácie bude obsahovať dve triedy, kde v jednej z tried budú alokované potrebné moduly a druhá trieda bude obsahovať metódy spojené s testovaním a dve aplikačné vlákna. Jedno aplikačné vlákno bude realizovať štart samotnej aplikácie spoločne s alokovaním a inicializáciou podporných modulov a druhé vlákno bude slúžiť na vlastné testovanie.

Testovacia aplikácia bude obsahovať testy na zbernice I2C a UART, sieťové komunikačné rozhrania WiFi a Ethernet a v neposlednom rade aj úložiska eMMC. Tieto

testy budú implementované v samostatných metódach, okrem zbernice UART, ktorá sa využíva na komunikáciu na MCU, čiže jeho funkčnosť je testovaná počas celého fungovania samotnej aplikácie.

Keďže testovací výrobok ESP32 v spoločnosti Elcom, je dostupný v dvoch variantoch modulov, tak rozdelenie tejto aplikácie do dvoch typov, nebude nutný. Toto rozdelenie bude aj vzhľadom k potrebe nastaviť pre každý prípravok samostatnú konfiguráciu MCU vyriešené formou podmienených prekladov. Ako bolo v analýze uvedené, funkčne sa jedná o rozdiel obsahu jedného testu naviac oproti tomu druhému typu modulu, čiže v jednom prípade, bude test spustiteľný a v druhom nie.

Nová testovacia aplikácie nespôsobí žiadne zmeny, ktoré by sa odrazili na procese samotného testovania v výrobe, to znamená, že postup, ktorý sa aplikoval do teraz nebude potrebné obmieňať a ani nie je potrebná žiadna zmena testovacích prípravkov.

3.2 Návrh architektúry aplikácie

3.2.1 Hlavná funkcia

Každý samostatný firmvér musí obsahovať funkciu, ktorá odštartuje jeho fungovanie. ESP32 predpisuje štandard, ktorý je potrebný dodržiavať aby bolo možné vyprodukovaný firmvér skompilovať a následne spustiť na konkrétnej platforme na konkrétnom jadre.

Hlavná funkcia tohto firmvéru bude app_main(), ktorej úlohou je spustiť testovaciu aplikáciu spoločne s potrebnými nastaveniami.

V prvom kroku tejto funkcie bude nastavenie logovania a to na zabezpečenie výpisu logov do konzoly pre informovanie používateľa, napríklad o tom, aký test sa práve spustil, aký je jeho výsledok alebo či nastala nejaká kolízia na strane firmvéru. Následne, po umožnení logovanie je výpis prvého logu, ktorý informuje o poslednom type reštartu. Táto informácia slúži na povedomie o tom, či predošlé vypnutie ESP32 nebolo spôsobené nejakou chybou. Po informácií ohľadom posledného typu reštartu nastane nastavanie potrebných pamäťových blokov ESP32. Všetky tieto kroky sú implementované v samostatných funkciách.

```
#include <Logging.h>
#include <RestartReason.h>
#include <esp_system.h>
#include <PreInitialization.h>

extern void SystemTestStart();

void app_main()
{
    esp_log_set_vprintf(LogUartAndFile);
    log_reset_reason();
    PreInitialize();

    ESP_LOGW("MAIN THREAD", "Before system start: %u", esp_get_free_heap_size());
    SystemTestStart();
    ESP_LOGW("MAIN THREAD", "After system start: %u", esp_get_free_heap_size());
}
```

Obrázok č. 4: app_main() funkcia

(Zdroj: vlastné spracovanie)

Posledným krokom tejto hlavnej funkcii, je spustenie SystemTestStart(), ktorá má za úlohu samotnú inicializáciu hlavnej triedy celého firmvéru a spustenia štartovacej metódy tejto triedy. Spustenie tejto funkcie je sprevádzané informáciami o veľkosti Heap pamäte vo forme logov do konzoly.

```
std::unique_ptr<SystemTest> systemTest = nullptr;

extern "C" void SystemTestStart()
{
    systemTest = std::make_unique<SystemTest>();
    systemTest->StartSystemTest();
}
```

Obrázok č. 5: SystemTestStart() funkcia

(Zdroj: vlastné spracovanie)

3.2.2 Triedy

Ako som spomínal v podkapitole, ktorej podstatou bolo predstaviť všeobecný návrh celkovej testovacej aplikácie tento firmvér bude obsahovať dve triedy, ktoré budú na seba nadväzovať a budú využívať podstatu a výhody objektovo-orientovaného programovania.

Prvá trieda, ktorá je považovaná za hlavnú je SystemTest. Táto trieda obsahuje všetko, čo je potrebné pre funkčnosť firmvéru.

SystemTest je teda aj poskytovateľom modulov pre samotnú hlavnú činnosť tohto firmvéru a tým je testovanie hardvérových komponentov.

V prvom kroku opisu jednotlivých modulov, kde budem opisovať jednotlivé časti, ktoré by som rád rozdelil do dvoch skupín z dôvodu, korektnejšieho pochopenia. Prvou skupinou sú moduly, ktorých rozhranie priamo súvisí a umožňuje činnosť testovacích metód a druhá skupina modulov sú také, ktoré zabezpečujú funkčnosť samotného hardvéru.

Pod prvou skupinou modulov, je možné nájsť inštancie tried ako je Network, tento modul nám poskytuje metódy pre prácu s náležitosťami sieťových technológií, teda potrebné pre testovanie WiFi a Ethernet modulov. Storage, ktorý svojou implementáciou zabezpečuje základné operácie so súbormi a tak isto základne kroky inicializácie eMMC pamäte, čiže bez tohto modulu by otestovania eMMC nebolo možné realizovať.

V druhej skupine sa nachádzajú Settings, ktorý je potrebný pre samotnú inicializáciu Storage a Network, no a ešte tam patrí SystemStatus a Network Event, ktoré sú súčasťou inicializačného procesu Network(u).

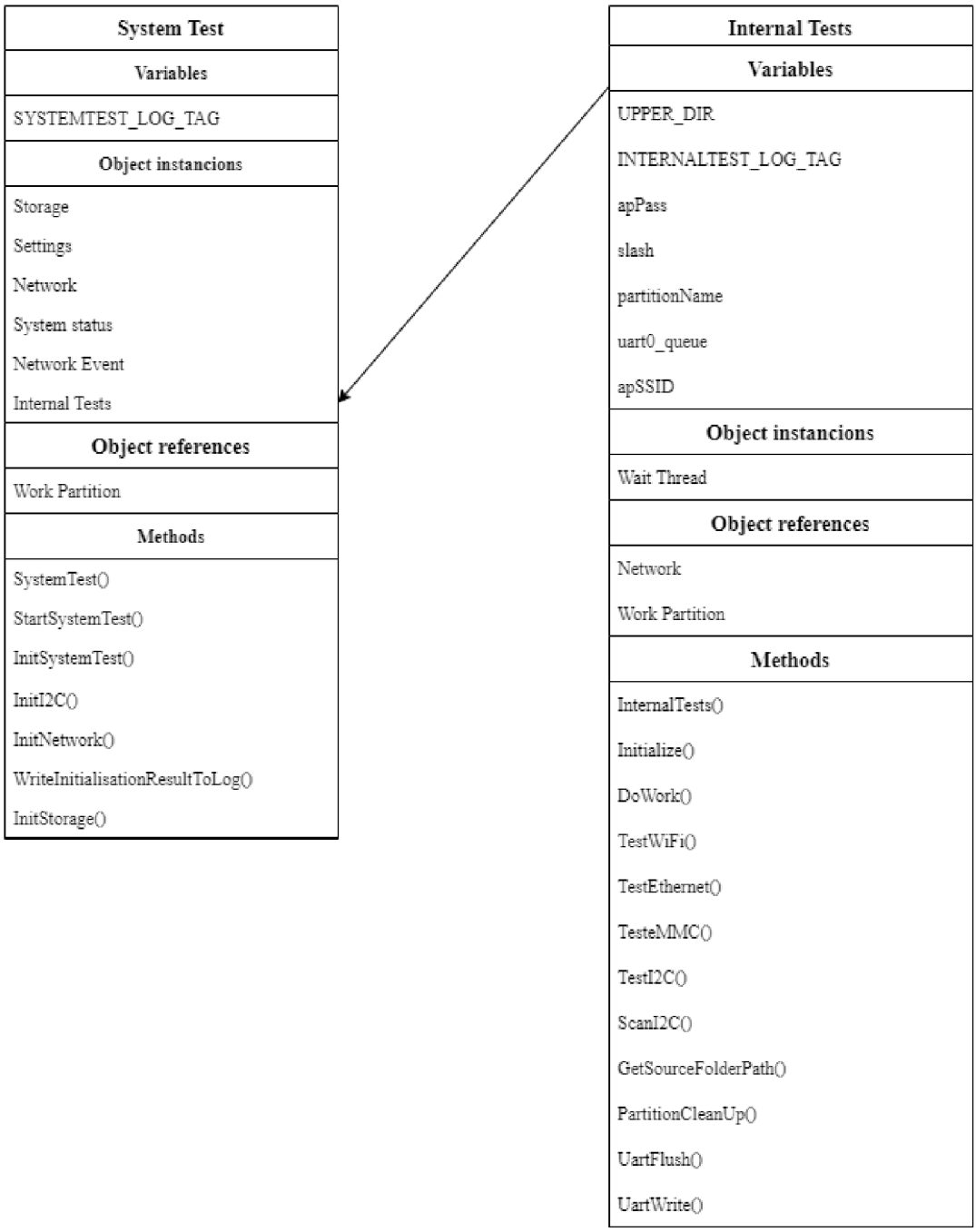
Trieda SystemTests neobsahuje len inštancie modulov týchto dvoch skupín modulov ale tak isto inštanciu samotného modulu pre testovanie jednotlivých komponentov MCU, kde sú implementované potrebné testovacie metódy a veci na zabezpečenie fungovania týchto metód.

Okrem inštancii modulov táto trieda obsahuje aj odkaz, na modul konkrétnej partície eMMC pamäte.

Druhou triedou tohto testovacie firmvéru je IntenalTests, ktorej podstatou je zabezpečenie funkcionality, spojenej so samotným vykonávaním testovania s všetkými náležitosti, ktoré si testovanie vyžaduje.

Trieda obsahuje inštanciu modulu pre prácu s vláknami operačného systému freeRTOS, či už sa jedná o samotné vytvorenie a spustenie vlákna alebo aj synchronizačné metódy, pre zabezpečenie vláknovej synchronizácie.

Nasledujúcou podstatnou častou tejto triedy je referencia na modul Network a ukazovateľ na partíciu v Storage, ktoré patria do prvej skupiny modulov, ktoré som spomínal v opise SystemTest triedy, teda moduly, ktoré slúžia na zabezpečenie rozhraní na vykonávanie samotných testov.



Obrázok č. 6: Bloková schéma tried

(Zdroj: vlastné spracovanie)

3.2.3 Príkazy

Táto testovacia aplikácie obsahuje dva rôzne príkazy. Jeden príkaz má názov TEST a druhým je príkaz SCAN. Názvy týchto príkazov predstavujú aj sekvenciu znakov potrebné na spustenie konkrétneho procesu testovacej aplikácie.

Spustenie týchto príkazov bude prebiehať cez samotný UART, ktorý bude prijímať vstupné údaje z externej periférie. Tú predstavuje napríklad klávesnica, kde používateľ zadá potrebnú sekvenciu znakov, na základe toho čo chce aby táto aplikácia vykonala.

V procese, kedy sa táto aplikácia bude využívať, teda pri ukončení procesu výroby modulu s ESP32, samotné spustenie bude možné na základe predpripraveného skriptu, ktorým počítač vo výrobnej hale disponuje a v princípe vykoná to, že zadá vstupnú sekvenciu príkazov aby túto činnosť nemusel vykonávať opakovane výrobný pracovník, teda zabezpečí spustenie oboch príkazov.

Príkaz TEST má za úlohu vykonať primárnu činnosť tejto aplikácie a tým je teda otestovať hardvérové komponenty samotného MCU. Teda po zadaní tejto sekvencii znakov, sa vykonajú jednotlivé test I2C, eMMC, WiFi a v prípade modulu typu WROVER-B aj test Ethernetu. Spojené s testovacími metódami bude aj informovanie o úspešnosti jednotlivých testov, zobrazené samostatne. Posledným krokom tohto príkazu je spustenie metódy na odstránenie podporných konfiguračných súborov, ktoré vznikli počas testovania.

Druhým a zároveň posledným príkazom je príkaz SCAN. Úlohou tohto príkazu je spustenie metódy ScanI2C. Tento príkaz predstavuje otestovanie funkčnosti testovacieho prípravku aby bolo zaručené korektné prostredie pre testovanie.

3.3 Aplikačná architektúra

V tejto kapitole sa zameriam na popis aplikačnej architektúry. Predstavuje popis jednotlivých vlákien aplikácie, proces spustenia, samotný chod celej aplikácie a realizáciu vstupných príkazov.

Celá testovacia aplikácia sa spúšťa a prebieha na samostatnom jadre, ktoré je aj určené pre aplikáciu MCU. Jedná sa o jadro APP_CPU, ktoré sa označuje indexom 1. Druhé

jadro, s indexom 0, zabezpečuje činnosť nízko-úrovňových komunikácii, napríklad pre I2C či UART.

Činnosť aplikácie je rozdelená do dvoch samostatných vlákien. Prvé vlákno predstavuje hlavné vlákno, ktoré sa spustí počas spustenia aplikácie, kde jeho úlohou je vykonať všetky inicializačné procesy jednotlivých modulov, teda I2C, Storage, Network a v poslednom kroku tohto vlákna je inicializácia konkrétneho UARTu pre zabezpečenie prijímania príkazov a zobrazenie výstupu. Takže vlastne pripravuje všetko tak, aby mohlo prebiehať testovanie korektne so všetkými svojimi požiadavkami a väzbami na moduly.

Druhé vlákno je testovacie vlákno, ktoré vykonáva testovacie procesy. Ich úlohou je okrem testovanie samozrejme aj poskytnutie výstupu, teda informácia o úspešnosti alebo neúspešnosti testovania. Testovacie vlákno sa spúšťa asynchrónne voči hlavému vláknu v inicializačnom procese testovacej triedy. Po vykonaní všetkých inicializačných krokov, ktoré sa dejú v hlavnom vlákne, toto vlákno skončí a na úrovni aplikačnej vrstvy ostane bežať už iba testovacie vlákno, keďže to jediné je potrebné na samotný chod firmvéru, keď sa už inicializačné procesy skončia.

Po ukončení všetkých inicializácií a spustení testovacieho vlákna, je všetko pripravené na vykonávanie testovania modulov. To znamená napríklad, že test pre WiFi, ktorý potrebuje využívať modul pre sieťové zariadenia ma k dispozícií všetky potrebné náležitosti aby vedel vykonávať svoju činnosť.

Aplikácia už teda beží iba v jenom vlákne teda, kde prebieha nekonečný cyklus, ktorý čaká na prijímania vstupných bytov, ktoré môžu prichádzať cez UART. To predstavuje vstup, z externých periférií, kde aplikácia očakáva presnú sekvenciu znakov, ktoré predstavujú konkrétny názov príkazu, napríklad TEST. To znamená, že používateľ zadá cez externú periférium, napríklad cez klávesnicu, znak po znaku požadovaný príkaz, UART ho príjme a prenesie do ESP32 a ten už má zadefinované čo má v danej situácii vykonať. V prípade, že používateľ zadáva nekorektné vstupné údaje tak to aplikácia vie vyhodnotiť, nekorektný vstup ako nesprávny. Keďže sa jedná o nekonečný cyklus, tieto operácie je možné vykonávať viac-násobne bez akýchkoľvek obmedzení.

Všetky informácie o priebehu chodu aplikácie, či už výstupy testovania alebo systémové oznámenia aplikácie sú pomocou UARTu vypísané do konzoly.

O ukončenie fungovania aplikácie sa stará samotný testovací prípravok, ktorý na základe svojich možností MCU vypne.

3.4 Hlavná trieda

Ako bolo spomínane, trieda SystemTest obsahuje inštancie a referencie na špecifické typy modulov, no nepredstavuje to celú podstatu a implementáciu tejto triedy. Okrem týchto spomínaných náležitostí, trieda definuje objektové metódy a objektovú premennú, ktoré sú rozdelené do konkrétnych sekcií z hľadiska ich prístupnosti. Teda ak sa jedná o obsah triedy, ktorá je súkromná, predstavuje možné použitie daných metód alebo inštancií v samotnom objekte. Pod druhým pojmom sa nachádzajú metódy, ktoré sú takzvane verejné, ich využitie je teda možné aj mimo daného objektu.

Trieda SystemTest obsahuje jednu premennú, ktorá slúži ako tag pre výpisy istých logov do konzoly, aby bolo možné identifikovať, odkiaľ daný log pochádza, teda z akého konkrétneho modulu pre lepšiu orientáciu.

Z hľadiska metód, sa tam nachádzajú také, ktoré sú zodpovedné za inicializácie modulov slúžiacich na chod tohto firmvéru a aj jedna podporná metóda.

Keďže tento firmvér je postavený na objektovo-orientovanom programovaní, tak okrem toho, že sú v ňom implementované triedy tak sa využíva ich možnosť pri inicializácii samotného objektu, vykonať špecifické operácie pomocou konštruktora.

Podstata, ktorá sa skrýva za konštruktorom triedy SystemTest je inicializácia konštruktorov ostatných modulov tejto triedy. Keďže tieto moduly vzájomne komunikujú, tak je potrebné zabezpečiť ich prepojenosť. Súčasne v tele konštruktora sa vykonávajú operácie spojené so základnými informáciami o firmvéri a MCU formou výpisu do konzoly. Nimi sú informácia o verzii esp-idf, firmvéru, jeho názve a verzii, ktorá je nahratá v MCU.

```
class SystemTest final
{
    public:
      static constexpr const auto SYSTEMTEST_LOG_TAG = "System Test";

    private:
        SharedResource<ProtectedStorage> storage;
        SettingsProvider                 settings;
        NetworkManagementModule          network;
        SystemStatus                     systemStatus  = {};
        Event<ENetworkManagementEvents>  watchNetworkEvent;
        InternalTests                    internalTests;

    private:
        ProtectedStoragePartition*       workPartition = nullptr;

    public:
        SystemTest()
                    :
                    storage         (settings),
                    settings        (storage),
                    network         (settings, systemStatus, watchNetworkEvent),
                    internalTests   (network)
        {
          const auto idfVersion = esp_get_idf_version();
          ESP_LOGI(SYSTEMTEST_LOG_TAG, "ESP-IDF Version: %s", idfVersion);

          const auto & firm = SystemConsts::System::Firmware;
          ESP_LOGI(SYSTEMTEST_LOG_TAG,
                    "System version: %s %s %s HW" TO_ASTRING_MACRO(_ICMCFG_ICM_HW_VERSION),
                    firm.ManufacturerName,
                    firm.FirmwareName,
                    firm.Ver.ToString().c_str());
        }

    public:
         void StartSystemTest();

    private:
        template<typename T>
        void WriteInitialisationResultToLog(const char* const initialisationName,
                                            const T Result);
        void InitSystemTest();
        esp_err_t InitI2C();
        void InitNetwork();
        bool InitStorage();
};
```

Obrázok č. 7: Trieda SystemTest

(Zdroj: vlastné spracovanie)

3.4.1 Inicializačné metódy

Prvou zo skupiny inicializačných metód, ktorou by som rád začal je InitStorage(). Táto metóda okrem toho, že vykonáva inicializáciu celého súborového systému samotného

MCU, tak prideľuje referenciu, na konkrétny ukazovateľ partície súborového systému, ktorá je potrebná na následnú činnosť v metódach testovacej triedy.

```
bool SystemTest::InitStorage()
{
    auto guardedStorage = this->storage.LockGuard();
    this->workPartition =
            &guardedStorage->Card.Partitions[static_cast<int>(PartitionType::Workspace)];
    return guardedStorage->Initialize();
}
```

Obrázok č. 8: InitStorage() metóda

(Zdroj: vlastné spracovanie)

V prvom kroku tejto funkcie sa vytvorí mutex, na zabezpečenie synchronizáciu procesov, následne spomínaná inicializácie ukazovateľa na partíciu a v poslednom kroku zavolanie metódy z externého modulu pre súborový systém pre celkovú inicializáciu. Návratová hodnoty typu bool udáva informáciu o úspešnosti tejto inicializácie.

Ďalšou metódou z tejto skupiny, je metóda InitNetwork() zodpovedná za inicializáciu všetkých sieťových prvkov, potrebných na testovanie či už WiFi, alebo aj Ethernetu.

```
void SystemTest::InitNetwork()
{
    this->network.Initialize();
}
```

Obrázok č. 9: InitNetwork() metóda

(Zdroj: vlastné spracovanie)

InitI2C() predstavuje metódu na zabezpečenie možnosti využívať I2C zbernicu s všetkými jej náležitosťami a poskytovanými. Tak isto nastavuje použité a nepoužívané piny MCU.

```
esp_err_t SystemTest::InitI2C()
{
    gpio_config_t io_confUnused;
    io_confUnused.intr_type = GPIO_INTR_DISABLE;
    io_confUnused.mode = GPIO_MODE_INPUT;
    io_confUnused.pin_bit_mask = _EMUTESTCFG_HAVE_UNUSED_GPIO_PINS;
    io_confUnused.pull_down_en = GPIO_PULLDOWN_ENABLE;
    io_confUnused.pull_up_en = GPIO_PULLUP_DISABLE;
    gpio_config(&io_confUnused);

    return i2c_init(I2C_MODE_MASTER);
}
```

Obrázok č. 10: InitI2C() metóda

(Zdroj: vlastné spracovanie)

Poslednou zo skupiny týchto inicializačných metód je InitSystemTest(). Jej úlohou je volanie inicializačných metód tejto triedy, teda zabezpečenie ich spustenia a zároveň informovanie o ich úspešnosti pomocou logov, no a okrem volania metód tejto triedy, tak isto volá priamo inicializačnú metódu samotnej testovacej triedy.

```
void SystemTest::InitSystemTest()
{
    const auto resultOfInit = this->InitI2C();
    this->WriteInitialisationResultToLog("I2C", resultOfInit);
    this->systemStatus.ProtectedStorageError = this->InitStorage();
    this->WriteInitialisationResultToLog("ProtectedStorage",
                                         systemStatus.ProtectedStorageError);
    this->InitNetwork();
    this->internalTests.Initialize(this->workPartition);
}
```

Obrázok č. 11: InitSystemTest() metóda

(Zdroj: vlastné spracovanie)

Prvým krokom je spustenie inicializácie I2C a získanie návratovej hodnoty tejto funkcie do premennej a následne predanie tejto premennej do metódy, ktorá informáciu o úspešnosti alebo možnej neúspešnosti premietne vo forme logov do konzoly. Po výpise tejto informácie nastáva volanie funkcie na inicializácie súborového systému a uloženie návratovej hodnoty do systémovej premennej, ktorú následne vloží do argumentu pre

spomínaný výpis do logov. Poslednými dvoma krokmi sú teda spustenie metód na inicializáciu sieťových prvkov a samotnej testovacej triedy, s parametrom ukazovateľa na partíciu eMMC úložiska.

3.4.2 Podporné metódy

V tejto skupine podporných metód, sa nachádzajú metódy, ktoré napomáhajú funkčnosti celej tejto testovacej aplikácie z hľadiska umožnenie spustenia inicializačných procesov alebo podporu výpisu logov do konzoly. V tomto prípade sa jedná o dve metódy, prvou je StartSystemTest(). Podstatou tejto metódy, je zabezpečenie spustenie InitSystemTest(), teda spustiť celý inicializačný proces.

```
void SystemTest::StartSystemTest()
{
    this->InitSystemTest();
}
```

Obrázok č. 12: StartSystemTest() metóda

(Zdroj: vlastné spracovanie)

Poslednou metódou tejto skupiny je WriteInitialisationResultToLog(). Jej podstatou je zabezpečenie možnosti výpisu logov do konzoly inicializačných procesov pre sprostredkovanie informácii o úspešnosti inicializácie.

```
template<typename T>
void SystemTest::WriteInitialisationResultToLog(const char* const initialisationName,
                                                const T Result)
{
    static constexpr const auto initText = "initialization";
    std::size_t messageSize = snprintf(nullptr, 0,"%s%s",initialisationName, initText) + 2;
    auto message = std::make_unique<char[]>(messageSize);
    snprintf(message.get(), messageSize, "%s %s", initialisationName, initText);
    if (std::is_same<T, bool>::value)
    {
        LogSuccessOrFailMessage(SYSTEMTEST_LOG_TAG, message.get(), Result);
    }
    else if (std::is_same<T, esp_err_t>::value)
    {
        LogOperationResult( SYSTEMTEST_LOG_TAG, message.get(), Result);
    }
    else
    {
        ESP_LOGE(SYSTEMTEST_LOG_TAG, "Error data type parameter !");
    }
}
```

Obrázok č. 13: WriteInitialisationResultToLog() metóda

(Zdroj: vlastné spracovanie)

Parametre danej metódy sú názov inicializačného modulu a výsledok tejto operácie. Následne prebieha vytvorenie vhodného buffera pre konečný výstup a zvolenie vhodnej logovacej operácie na základe typu parametra výsledku inicializačného procesu.

3.5 Testovacia trieda

Trieda InternalTests už priamo v sebe deklaruje metódy, ktoré odrážajú primárnu funkcionalitu danej aplikácie. Okrem týchto metód, sa tu nachádzajú aj inštancie, referencie a ukazovatele, ktoré boli spomínane už v predošlej kapitole týkajúcej sa návrhu architektúry.

Konštruktor tejto triedy, zabezpečí priradenie referencie pre Network. Z hľadiska premenných sa tam nachádza širšia škála. V prvom rade sú to pomocné premenné pre podporné metódy PartitionCleanUp() a GetSourceFolderParh(). Prvá premenná definuje fixný názov partície a druhá koreňový adresár a slúži, pre funkciu IsUpperDirectory(). Potom tam sú premenné, ktoré sú používané pri testovaní WiFi, definujú SSID a heslo

prístupového bodu vo výrobnom procese. Poslednou častou sú premenné pre prácu s UARTom, teda port UARTu a popisovač pre UART frontu, ktoré sú potrebné pre inicializáciu UARTu a pre získavanie dát, ktoré prichádzajú z externého vstupu, ktorý zadáva používateľ tejto aplikácie.

```
class InternalTests final
{
  public:
    static constexpr auto UPPER_DIR = "..";

  private:
    static constexpr const auto INTERNALTEST_LOG_TAG = "Internal Tests";
    static constexpr auto apSSID = "TestICM";
    static constexpr auto apPass = "xm2tobav";
    static constexpr auto slash = "/";

    const std::string partitionName = "/Work";

    static constexpr uart_port_t port = UART_NUM_0;

    QueueHandle_t uart0_queue = {};
    Threading::Thread waitThread;
    NetworkManagementModule &network;
    ProtectedStoragePartition *workPartition = nullptr;

  public:
    InternalTests(NetworkManagementModule &v_network):
        network(v_network)
    {}

  public:
    void Initialize(ProtectedStoragePartition *WorkPartition);

  private:
    void DoWork();
    void TestWiFi();
    void TesteMMC();
    void TestI2C();
    void ScanI2C();
    std::string GetSourceFolderPath(const std::string& path) const;
#ifdef _EMUTESTCFG_HAVE_ETHERNET_MODULE
    void TestEthernet();
#endif  //_EMUTESTCFG_HAVE_ETHERNET_MODULE
    void PartitionCleanUp(const std::string& path);
    void UartFlush();
    void UartWrite(const std::string& text);
};
```

Obrázok č. 14: Trieda InternalTests

(Zdroj: vlastné spracovanie)

3.5.1 Inicializačná metóda

InternlTests trieda obsahuje jednu inicializačnú metódu, ktorá ma za úlohu priradenie hodnoty ukazovala pre prácu s partíciou, ďalej vykonať inicializáciu UARTu na požadovanú prácu, teda aby bolo možné získavať dáta z externej periférií a následne s nimi pracovať a teda tým aj zabezpečiť otestovanie UARTu a na koniec spustiť testovacie vlákno.

```
void InternalTests::Initialize(ProtectedStoragePartition* WorkPartition)
{
    this->workPartition = WorkPartition;
    uart_driver_install(port, 200, 200, 20, &uart0_queue, 0);
    uart_enable_pattern_det_intr(port, '+', 1, 10000, 1000, 1000);
    uart_pattern_queue_reset(port, 20);
    waitThread.Start([this]() { this->DoWork(); },
                     Threading::ThreadPriority::BelowRealtime,
                     4096, "InternalTest", 1);
}
```

Obrázok č. 15: Initialize() metóda

(Zdroj: vlastné spracovanie)

Parametrom tejto metódy je samotný ukazovateľ na konkrétnu partíciu, ktorý mieri na požadovaný modul. V prvom kroku vykonávania tejto metódy, sa táto hodnota z parametra skopíruje do objektovej premennej, v nasledujúcich krokoch nastáva inicializácie samotného UART ovládača, nakonfigurovanie na možnosť získavania dát z externej periférií, čiže v tomto firmvéri na prijímanie spomínaných príkazov a v poslednom kroku procesu inicializácie UARTu je alokovanie konkrétnej veľkosti pamäte pre prácu s frontom. Poslednou úlohou danej metódy je spustenie testovacieho vlákna, s konkrétnou metódou, ktorá sa bude vykonávať spoločne s aktívneho referenciou objektu, určením systémovej priority vlákna, veľkosť zásobníka, pomenovanie vlákna a indexu CPU, kde bude danú operáciu vykonávať.

3.5.2 DoWork() metóda

Metódu DoWork() je možno označiť za výkonnú, za nositeľa celkovej funkcionality samotného vykonávanie testovacích procesov a spracovateľa príkazov. Jej úlohou je spracovať daný príkaz, či už sa jedná o SCAN alebo TEST a na základe tejto informácie

spustiť požadovanú sekvenciu metód na vykonanie definovaného procesu. Ďalšou podstatnou častou je nastavenie potrebných náležitosti pre samotný test WiFi, ktoré je potrebné vykonať pred spustením procesov.

```
void InternalTests::DoWork()
{
    static constexpr auto testText = "TEST"sv;
    static constexpr auto scanText = "SCAN"sv;
    uint8_t writePos = 0;
    uart_event_t event = {};

    uint8_t data[20] = {0};
    esp_log_level_set("*", ESP_LOG_NONE);
    network.WiFiModule.setSsid(this->apSSID);
    network.WiFiModule.setPassword(this->apPass);
    network.WiFiModule.SetEnableSTA(true);
    vTaskDelay(pdMS_TO_TICKS(5000));
    network.WiFiModule.Reconnect();

    this->UartWrite("TEST_PROCEDURE_INITIALIZED\r\n");
    for (;;)
    {
        if (xQueueReceive(uart0_queue, (void*)&event, (portTickType)portMAX_DELAY))
        {
            if (event.type == UART_DATA)
            {
                writePos += uart_read_bytes(port, &data[writePos],
                            std::min(sizeof(data) - writePos, event.size), (portTickType)portMAX_DELAY);
                if (std::islower(data[writePos - 1]) != 0)
                {
                   data[writePos - 1] = std::toupper(data[writePos - 1]);
                }
            }
            auto swData = string_view(reinterpret_cast<char*>(data), writePos);
            if ((writePos >= testText.length()) && (swData == testText)) // test command
            {
                vTaskDelay(pdMS_TO_TICKS(3000)); // waiting for drivers are ready, no action is required
                this->UartWrite("\r\nBEGIN\r\n");
                this->TestI2C();
                this->TesteMMC();
                this->TestWiFi();
#ifdef _EMUTESTCFG_HAVE_ETHERNET_MODULE
                this->TestEthernet();
#endif  //_EMUTESTCFG_HAVE_ETHERNET_MODULE
                this->UartWrite("END\r\n\032");
                this->PartitionCleanUp(this->partitionName);
                writePos = 0;
            }
            if ((writePos >= scanText.length()) && (swData == scanText)) //scan command
            {
                esp_log_level_set("*", ESP_LOG_INFO);
                vTaskDelay(pdMS_TO_TICKS(3000));
                this->UartWrite("\r\nBEGIN\r\n");
                this->ScanI2C();
                this->UartWrite("END\r\n\032");
                writePos = 0;
                esp_log_level_set("*", ESP_LOG_NONE);
            }
            if ((writePos >= (testText.length() + 1)) // input check
                || (writePos >= (scanText.length() + 1))
                || (!std::isalpha(data[writePos - 1])))
            {
                this->UartFlush();
                writePos = 0;
            }
            vTaskDelay(pdMS_TO_TICKS(100)); // no action is required
        }
    }
}
```

Obrázok č. 16: DoWork() metóda

(Zdroj: vlastné spracovanie)

Na začiatku metódy nastáva vytvorenie potrebných premenných, ktoré reprezentujú názov príkazov tejto aplikácie a potom premenných pre spracovanie vstupov, či už sa jedná o buffer, index buffera a UART event.

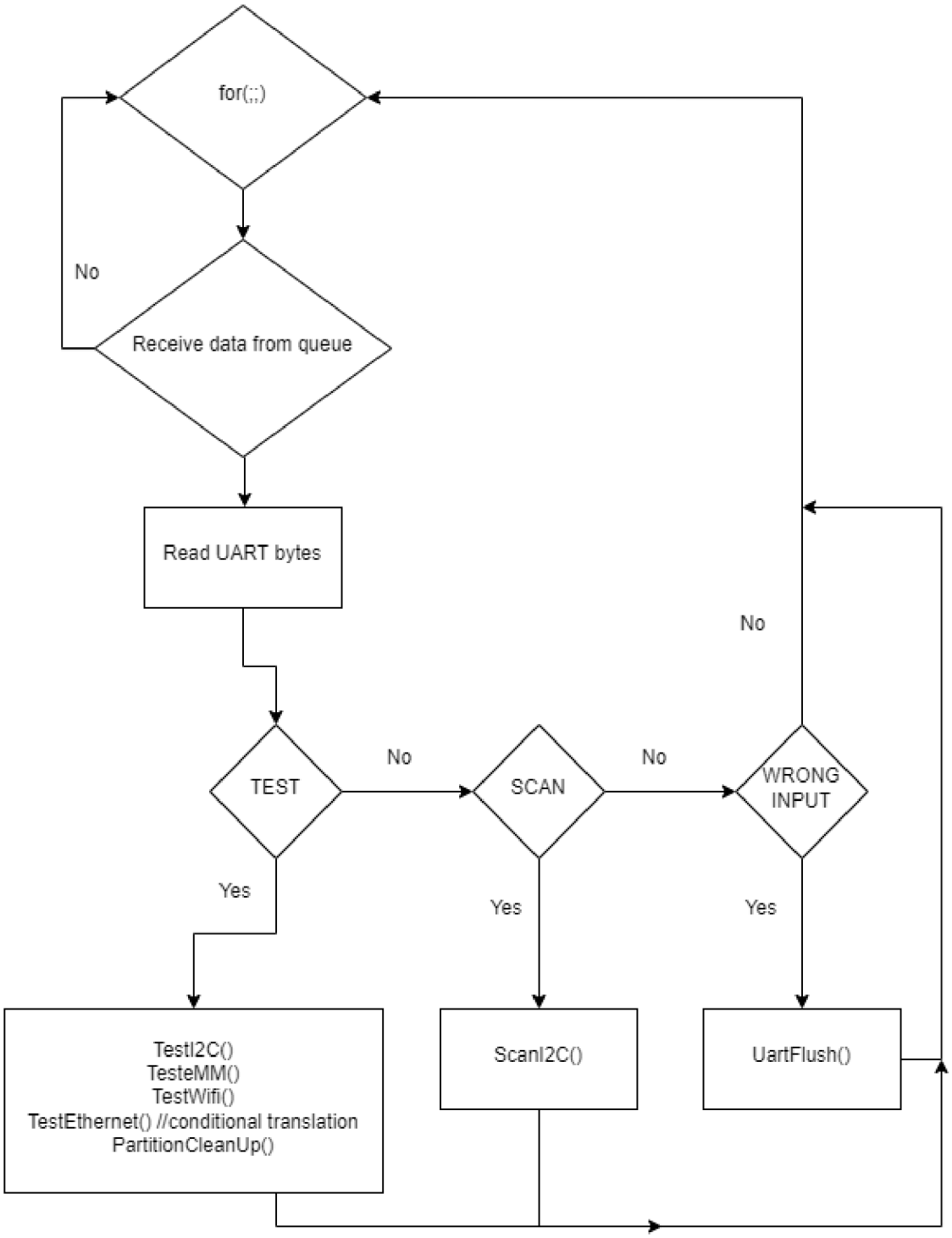
Následne vypnutie interných logov, keďže v priebehu vykonávania testovacích operácii nie je vhodné, aby boli zapnuté a zasahovali do výstupu testu.

Potom prebieha nastavenie SSID a hesla AP vo výrobnej hale, nastavenie ESP32 ako klienta, teda pre umožnenie sa pripojiť na AP. Po vykonaní týchto operácii pozastavenie práca testovacieho vlákna na 5 sekúnd aby všetky tieto operácie boli schopné sa vykonať a následne vykonať operáciu opätovného pripojenia.

Po vykonaní týchto operácii ohľadom WiFi nastane UART výpis do konzoly s informáciou o inicializácií testovacieho modulu a spustenie nekonečného cyklu, v ktorom nastáva možnosť získavania dát z UART fronty, ktorá sa napĺňa alebo teda získava dáta, ktoré sú zadané z vstupu, ktorý je pripojený na UART.

Ak UART front obsahuje niečo, tak nastane vyčítanie daných bytov do buffera, následne kontrola či daný byte je v správnej forme, ak nie, tak nastane jeho oprava na veľký znak. Po tejto úprave, sa daný buffer spoločne s jeho dĺžkou upraví do jednotného konceptu, konkrétne do premennej typu string_view aby následne bolo možné vykonávať s týmto už reťazcom operácie, slúžiace na zistenie o aký príkaz sa jedná a na základe toho spustiť už konkrétne procesy.

V prípade, že vstupný reťazec nevyjadruje žiaden z možných príkazov a jeho dĺžka prekročí dĺžku povoleného reťazca nastane vyčistenie UART fronty.



Obrázok č. 17: Diagram pre metódu DoWork()

(Zdroj: vlastné spracovanie)

3.5.3 Test I2C

Úlohou metódy TestI2C() je otestovať rozhranie I2C a podať informáciu o výstupe testu, čiže o tom, či je I2C v poriadku a je schopná produkčného používania. Funkcionalita tejto metódy je postavená na tom, že náhodne vygenerované hodnoty, pomocou I2C zapíše do externej EEPROM pamäte, následne tieto hodnoty z pamäte vyčíta tak isto pomocou I2C a porovná či vstupné hodnoty sa rovnajú výstupným, teda že I2C ich dokázala zapísať a aj korektne v tom istom poradí vyčítať. Ak sú hodnoty rovnaké, test sa vyhodnotí úspešne čo bude svedčiť o korektnej funkčnosti I2C.

```
void InternalTests::TestI2C()
{
    static constexpr const auto TESTING_SIZE = 16;
    static constexpr const auto EEPROM_PAGE_SIZE = 8;
    uint8_t starting_address = 0x00;
    esp_err_t result = ESP_FAIL;
    uint8_t page_write_data[TESTING_SIZE] = {0};
    uint8_t page_read_data[TESTING_SIZE] = {0};
    uint8_t EEPROM_ADDRESS{};

    // Reset FIFO buffers
    i2c_resetRxFifo();
    i2c_resetTxFifo();
    esp_fill_random(page_write_data, TESTING_SIZE);
    do
    {
        result = i2c_scanAddress(&EEPROM_ADDRESS, true);
        if (result != ESP_OK) break;
        // Write to EEPROM
        result = i2c_writeDeviceAsMaster(page_write_data, TESTING_SIZE,
                                         EEPROM_ADDRESS, starting_address, EEPROM_PAGE_SIZE);
        if (result != ESP_OK) break;
        // Read from EEPROM
        result = i2c_readDeviceAsMaster(page_read_data, TESTING_SIZE,
                                        EEPROM_ADDRESS, starting_address, EEPROM_PAGE_SIZE);
        if (result != ESP_OK) break;
        for (int i = 0; i < TESTING_SIZE; i++)
        {
            if (page_write_data[i] != page_read_data[i])
            {
                result = ESP_FAIL;
                break;
            }
        }
    } while(0);

    if (result == ESP_OK)
    {
        this->UartWrite( "I2C: PASS\r\n" );
    }
    else
    {
        this->UartWrite( "I2C: FAILED\r\n" );
    }
}
```

Obrázok č. 18: TestI2C metóda

(Zdroj: vlastné spracovanie)

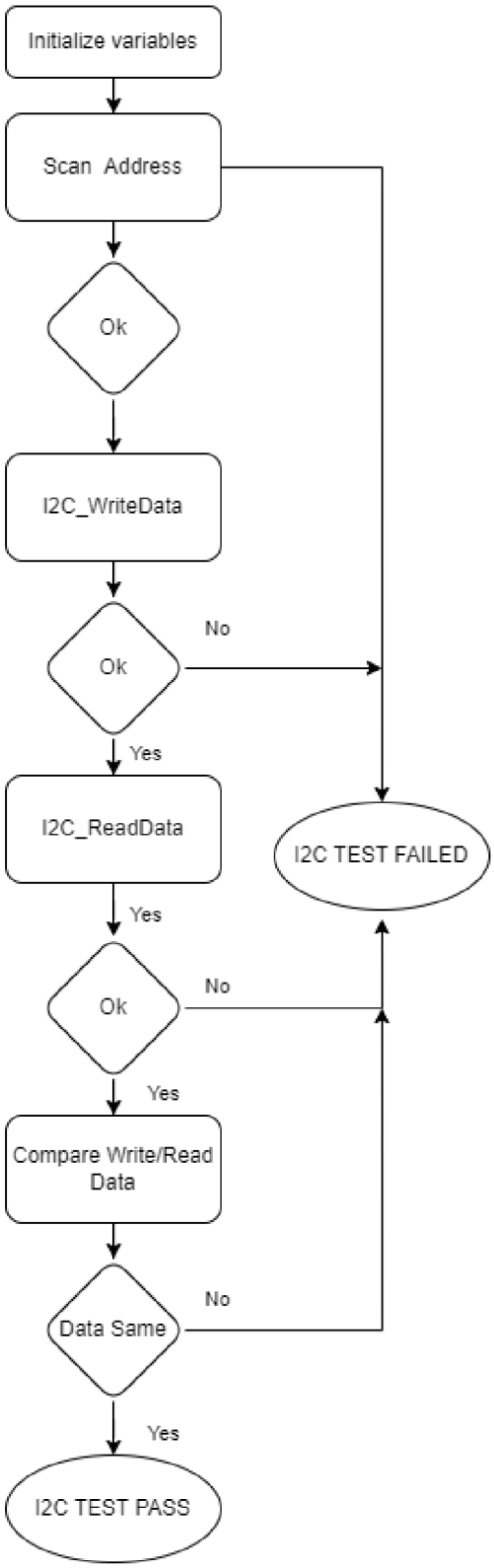
V prvom kroku nastáva vytvorenie premenných. Prvá určuje veľkosť buffera, ktorý bude zapísaný do EEPROM a druhá množstvo bytov, ktoré je možné zapísať/vyčítať v jednom cykle zápisu/čítania pomocou I2C do EEPROM pamäte.

V ďalšom kroku je vytvorenie premennej pre výsledok operácii, potom počiatočnej adresy pre zápis prvého bytu, polia obsahujúce dáta pre zápis a čítanie dát do/z EEPROM cez I2C. Poslednou premennou je adresa, na ktorej vie I2C komunikovať priamo s EEPROM, ktorá sa inicializuje až neskôr.

Prvé funkcie, ktoré sú vykonávane je resetovanie I2C bufferov, potom nastane vygenerovanie náhodných hodnôt, ktoré sú využiteľné ako dáta pre zápis. Po tom, čo sa obdržia tieto hodnoty, nastane zistenie adresy, na ktorej EEPROM a I2C spoločne vedia komunikovať a vykonávať procesy, ktoré nasledujú po tom a to je zápis a čítanie týchto dát cez I2C z EEPROM pamäte.

Po ukončení všetkých týchto činností nasleduje porovnanie dát, ktoré boli vytvorené na začiatku a následne sa s nimi pracovalo cez I2C ako spomínaný zápis a čítanie.

Ak sa tieto dáta rovnajú, teda I2C bola schopná ich zapísať a vyčítať, sa I2C považuje za otestovanú s kladným výstupom, teda je pripravená na fungovanie v produkčnom prostredí.



Obrázok č. 19: Diagram testu I2C

(Zdroj: vlastné spracovanie)

3.5.4 Test eMMC

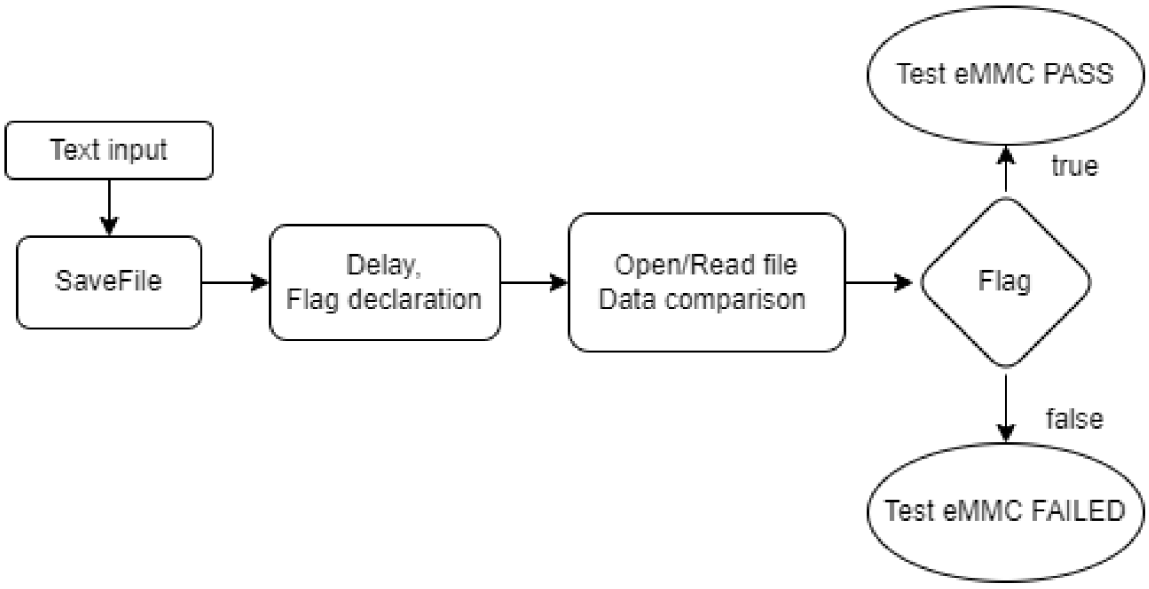
Úlohou tohto testu je otestovať korektnosť a fungovanie úložiska eMMC. V danej metóde sa odohráva vytvorenie súboru s textom TEST a následne otvorenie novo-vytvoreného súboru, kde sa vyčíta daný text a následne porovná. V prípade, že text je totožný s vstupným je, možné považovať test za úspešne vykonaný.

```
void InternalTests::TesteMMC()
{
    constexpr auto testText = "TEST"sv;
    WorkspaceFile::Save(WorkspaceFile::Test, testText);
    vTaskDelay(pdMS_TO_TICKS(2000));
    bool textEquals = false;
    WorkspaceFile::ProcessTextFile(
        WorkspaceFile::Test,
        [&textEquals, &testText](std::string const&  checkText)
        {
            textEquals = (checkText == testText);
        });

    if (textEquals)
    {
        this->UartWrite("eMMC: PASS\r\n");
    }
    else
    {
        this->UartWrite("eMMC: FAILED\r\n");
    }
}
```

Obrázok č. 20: TesteMMC() metóda

(Zdroj: vlastné spracovanie)

Prvým krokom danej metódy, je vytvorenie premennej, ktorá obsahuje text, ktorý bude zapísaný do súboru. Následným krokom je vytvorenie súbor a uloženie do eMMC. Zápis do EEPROM si vyžaduje dostatočný čas na uloženie údajov, preto nastane uspanie vlákna na dve sekundy a po ubehnutí konkrétneho časového intervalu prebehne vytvorenie príznaku a vyčítanie toho súboru a porovnanie vyčítaného obsahu súboru. Ak text je rovnaký, teda operácie uloženia a vyčítanie prebehla úspešne, tak príznak naberie hodnotu true a následná podmienka vyhodnotí test ako úspešný. V opačnom prípade, ak niektorá z operácii zlyhala, tak príznak naberie hodnotu false a daný test bude považovaný za neúspešný.



Obrázok č. 21: Diagram testu eMMC

(Zdroj: vlastné spracovanie)

3.5.5 Test WiFi

Primárnym a jediným účelom tohto testu je otestovanie jedného z komponentov sieťových prvkov, a to konrétne WiFi. Prvý krok, ktorý je potrebný na vykonanie práce s WiFi, bol vykonaný v metóde DoWork, kde bolo aj popísané čo sa vykoná.

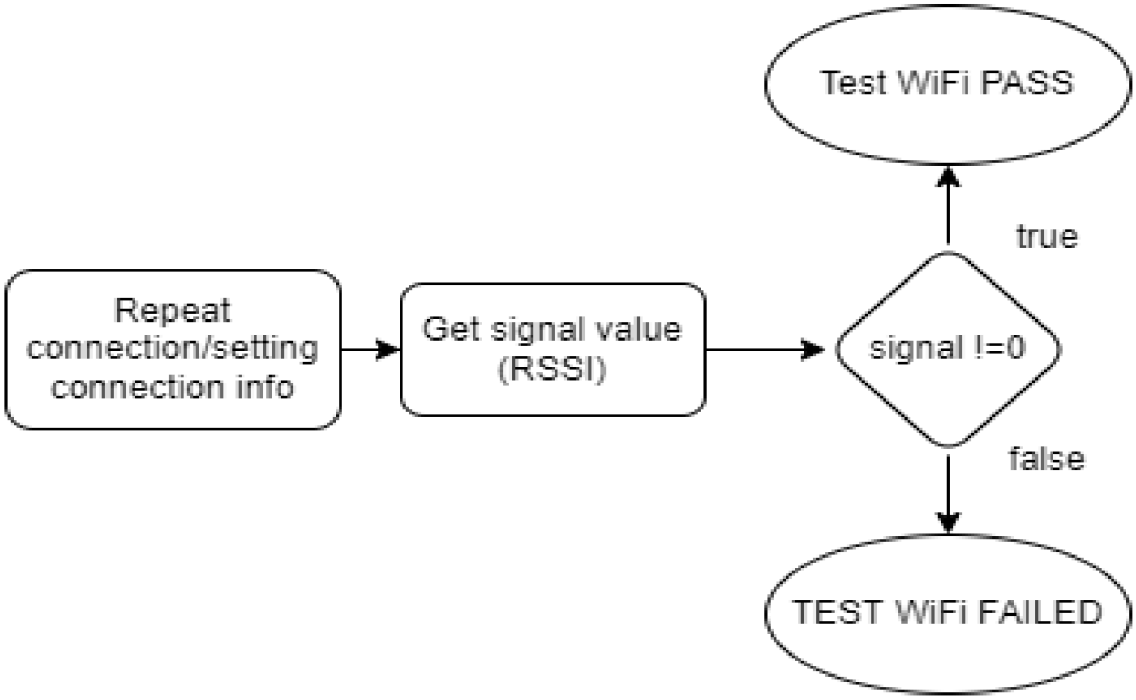
V testovacej metóde je už potrebné iba získať údaje o danom pripojení k AP v výrobnej hale a následne na základe týchto hodnôt posúdiť funkčnosti WiFi modulu. Konkrétne v tomto teste sa pracuje s vyčítanou úrovňou signálu vysielača. V prípade ak signál sa rovná nule je možné povedať, že WiFi modul sa nepripojil k AP a teda jeho funkčnosť je nesprávna. V prípade správneho pripojenia hodnota signálu má nenulovú hodnotu.

```
void InternalTests::TestWiFi()
{
    network.WiFiModule.WaitForConnection();

    const auto signal = network.WiFiModule.GetRSSI();

    if (signal != 0)
    {
        this->UartWrite(String::Format("WiFi: PASS, Signal: %i\r\n", signal));
    }
    else
    {
        this->UartWrite("WiFi: FAILED\r\n");
    }
}
```

Obrázok č. 22: TestWiFi() metóda

(Zdroj: vlastné spracovanie)

Metóda na začiatku vykonáva opätovné pripojenia ESP32 k AP, v definovanom počte pokusov a nastavuje hodnoty, ktoré boli po pripojení získané konkrétne napríklad IP adresa, jej maska alebo samotná úroveň signálu. Po vykonaní tejto operácia sa hodnota signálu vyhodnocuje a informuje používateľa o výsledku testu.



Obrázok č. 23: Diagram testu WiFi

(Zdroj: vlastné spracovanie)

3.5.6 Test Ethernet

Test Ethernet slúži na otestovanie Ethernet modulu ESP32. Na tomto mieste znova pripomínam, že ethernet je možné otestovať len pri module ESP32-WROVER-B. To je z dôvodu, že iný typ modulu neobsahuje možnosť Ethernet pripojenia.

Samotná metóda vykonáva kontrolu pripojenie a pridelenie IP adresy. Pokiaľ je možné zistiť IP adresu, tak sa test považuje za úspešný v opačnom prípade, keď sa IP adresa nepridelila, tak je možné považovať ethernet modul za chybný.

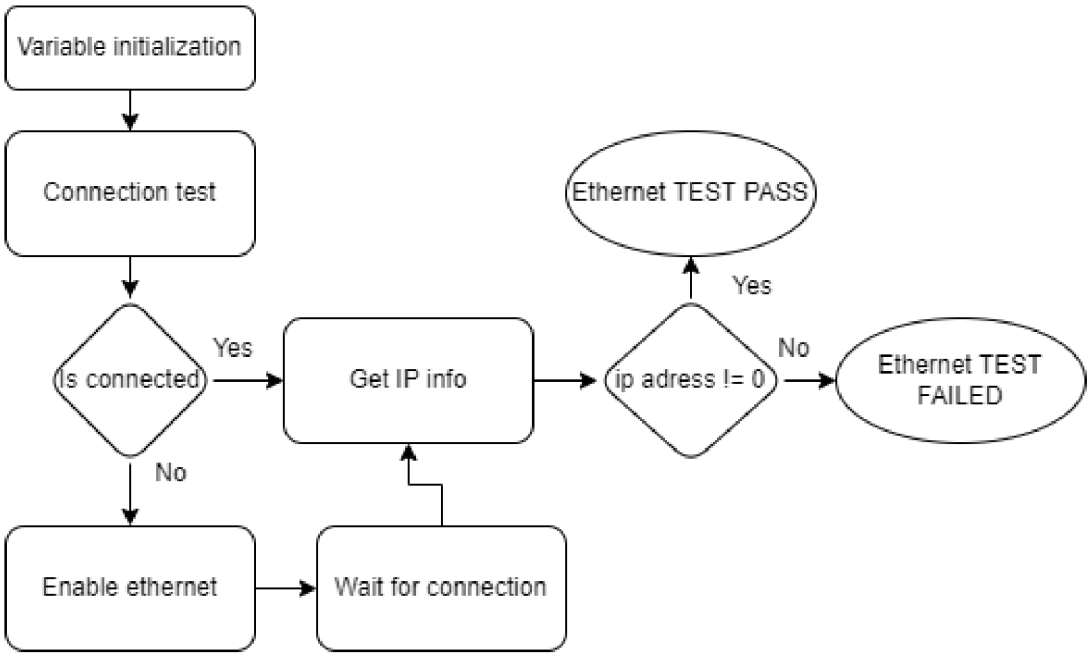
```
#ifdef _EMUTESTCFG_HAVE_ETHERNET_MODULE
void InternalTests::TestEthernet()
{
    tcpip_adapter_ip_info_t ipInfo;
    bool connected = network.IsConnected();
    if (!connected)
    {
        network.EthernetModule.Enable(true);
        network.EthernetModule.WaitForConnection();
    }
    ESP_ERROR_CHECK(tcpip_adapter_get_ip_info(TCPIP_ADAPTER_IF_ETH, &ipInfo));
    if (ipInfo.ip.addr != 0)
    {
        this->UartWrite("Ethernet: PASS\r\n");
    }
    else
    {
        this->UartWrite("Ethernet: FAILED\r\n");
    }
    if (!connected)
    {
        network.EthernetModule.Enable(false);
    }
}
#endif  //_EMUTESTCFG_HAVE_ETHERNET_MODULE
```

Obrázok č. 24: TestEthernet() metóda

(Zdroj: vlastné spracovanie)

Pri spustení tejto metódy v prvom kroku dochádza k vytvoreniu štruktúry, ktorá predstavuje všetky informácie ohľadom IP adresy. Následne nastane kontrola, či ESP32 je pripojené k zdroju internetu, v prípade negatívnej odpovede, to nemusí automaticky znamenať, že modul funguje nekorektne. Je možné, že na úrovni MCU Ethernet nebol povolený, preto v takomto prípade povolíme fungovanie ethernetu a opätovné čakáme na pokus o pripojenie. Tie je znova potrebné isté čakanie aby ovládače mali dostatočný čas na spracovanie informácií. Po týchto operáciách sa vykoná metóda na zistenie pridelenej

IP adresy, ak spomínaná štruktúra obsahuje nejakú konkrétnu IP adresu, test je vyhodnotený ako úspešný. Ak ani po týchto operáciách s povolením ethernetu a čakaním na pripojenie, IP adresa nebola pridelená, je možné tento modul považovať za nefunkčný a nevhodný na používanie. V prípade dodatočného povolenia Ethernet, ho dáme naspäť do pôvodného stavu, teda do stavu kedy jeho používanie nie je povolené.



Obrázok č. 25: Diagram testu Ethernet

(Zdroj: Vlastné spracovanie)

3.5.7 I2C Scan

ScanI2C() je metóda, ktorej využitie je zistiť na akej adrese komunikuje I2C s pamäťou EEPROM. Táto metóda sa využíva aj pri testovaní I2C. No primárne využitie spadá pod príkaz SCAN.

```
void InternalTests::ScanI2C()
{
    esp_err_t Result = ESP_FAIL;

    uint8_t Addresses[128] = {0};
    Result = i2c_scanAddress(Addresses, false);

    if (Result == ESP_OK)
    {
        for (int i = 0; i < sizeof(Addresses); i++)
        {
            if (Addresses[i] != 0)
            {
                this->UartWrite(
                    String::Format("I2C bus communicateson 0x%x address\r\n", Addresses[i]));
            }
        }
    }
    else
    {
        this->UartWrite("I2C scan did not find any devices on i2c bus ! \r\n");
    }
}
```

Obrázok č. 26: ScanI2C() metóda

(Zdroj: vlastné spracovanie)

Na začiatku metódy nastáva vytvorenie premennej na zachytenie výsledku nasledujúcej operácie a poľa, ktoré bude obsahovať konkrétne adresy, na ktorých I2C vie komunikovať s EEPROM.

Funkcia i2c_scanAddress poskytne možné adresy komunikácie. V prípade, že táto funkcia vráti návratovú hodnotu ESP_OK vieme, že na nejakej adrese sa podarila úvodná komunikácia a používateľ bude v logoch o tom informovaný. V prípade opaku, že sa nepodarilo komunikovať na nejakej z možných adries, usudzujeme, že testovací prípravok nie je schopný testovacieho procesu a tým pádom testy, ktoré by sa vykonali na takom to zariadení strácajú relevanciu.

3.5.8 PartitionCleanUp metóda

Metóda PartitionCleanUp slúži na odstránenie podporných súborov, ktoré boli vytvorené počas realizácie testovacích metód. Opodstatnenie tejto metódy je veľmi vysoké, z dôvodu toho, že je potrebné zabezpečiť kompletne prázdne úložisko po vykonaní testovacieho procesu.

Bez využívania metódy PartitionCleanUp, by nastala situácia, že po uvedenia samotného výrobku do používania, by stále obsahovalo súbory, ktoré nemajú nič spoločné s jeho funkčnosťou a tak isto by zaberalo nadbytočné miesto v pamäti úložiska eMMC.

Konkrétne sa jedná o súbory vytvorené pri testovaní eMMC a pri potrebných nastaveniach pre otestovanie WiFi. Tieto súbory sú uložené v svojej adresárovej štruktúre na jednej z partícií eMMC úložiska.

Spustenie tejto metódy nastáva v samotnom príkaze TEST ako jeho posledný krok s parametrom, ktorý predstavuje názov partície.

Prvým krokom metódy PartitionCleanUp je vyčítanie obsahu partície s parametrom, názvu konkrétnej partície do premennej, ktorá obsahuje názov súboru alebo adresára, spoločne s veľkosťou týchto položiek. Následne kontrola v podmienke, či vyčítanie bolo úspešne, v prípade neúspechu, bude používateľ upozornený formou výpisu do konzoly, že nastal problém pri tejto konkrétnej činnosti a samotná metóda sa ukončí.

Po kontrole, v predpokladanom stave, že všetko prebehlo úspešne, sa v cykle pole s obsahujúcimi informáciami o partícií rozbalí a prebieha následná kontrola, či sa jedná o adresár alebo súbor, no zároveň sa kontroluje, či sa nejedná o adresár, ktorý smeruje do koreňového adresára.

V prípade adresára, nastáva rekurzívne volanie PartitionCleanUp() s novým parametrom a to s konkrétnou cestou do toho adresára, aby bolo možné sa dopracovať k súborom.

Pokiaľ sa dostane na úroveň súborov, vykoná upravenie cesty k súboru a následne zavolá metódu, ktorá vykoná odstránenie súboru.

```
void InternalTests::PartitionCleanUp(std::string const& path)
{
    const auto dirContent = Directory::ReadDirectoryContent(path);

    if (::IsDirectoryContentError(dirContent))
    {
        esp_log_level_set("*", ESP_LOG_ERROR);
        ESP_LOGE(INTERNALTEST_LOG_TAG, "Read directory content failed");
        esp_log_level_set("*", ESP_LOG_NONE);
        return;
    }

    for (auto const& items : dirContent)
    {
        if (::IsDirectory(items) && !::IsUpperDirectory(items))
        {
            this->PartitionCleanUp(path + this->slash + items.first);
        }
        else if (!::IsUpperDirectory(items))
        {
            auto folderName = this->GetSourceFolderPath(path);
            this->workPartition->DeleteFile(folderName, items.first);
        }
    }
}
```

Obrázok č. 27: PartitionCleanUp() metóda

(Zdroj: vlastné spracovanie)

3.5.9 Podporné metódy testovacej triedy

Prvá skupina podporných metód testovacej triedy, slúžia na podporu funkcionality metódy PartitionCleanUp() a zabezpečenie prehľadnosti zdrojového kódu, pre jednoduchšiu orientáciu.

IsDirectoryContentError() slúži na kontrolu úspešnosti činnosti čítania obsahu partícii. IsDirecotory(), odpovedá na otázku či sa jedná o adresár, IsUpperDirecotry vyhodnocuje, či konkrétny adresár smeruje do koreňového adresára a poslednou z týchto podporných metód, je metóda GetSourceFolderPath(), ktorá upravuje cestu, pre mazanie súborov z dôvodov, že metóda, ktorá maže súbory, ma presné požiadavky pre svoje vstupné parametre.

```
static bool IsDirectoryContentError(const DirectoryList& dir)
{
    return (!(dir.at(0).first.empty()) &&
            (dir.at(0).second.compare(IDirectoryList::ERROR_PARAM) == 0));
}

static bool IsDirectory(const DirectoryEntry& item)
{
    return (item.second.compare(IDirectoryList::DIRECTORY_PARAM) == 0);
}

static bool IsUpperDirectory(const DirectoryEntry& item)
{
    return (item.first.compare(InternalTests::UPPER_DIR) == 0);
}

std::string InternalTests::GetSourceFolderPath(const std::string& path) const
{
    if (path.length() > this->partitionName.length() + 1)
    {
        return path.substr(this->partitionName.length() + 1,
                           path.length() - this->partitionName.length() + 1);
    }
    return this->slash;
}
```

Obrázok č. 28: Podporné metódy

(Zdroj: vlastné spracovanie)

Druhou skupinou týchto podporných metód je UartWrite0, ktorý sa využíva ako výpis do konzoly priamo cez testovaný UART, ktorý sa nastaví na základe čísla portu. No a ďalšou metódou je UartFlush(), ktorú využívame na odstránenie všetkých dát, ktoré sa nachádzajú v buffery testovacieho UARTU.

```
void InternalTests::UartWrite(const std::string& text)
{
    uart_write_bytes(port, text.data(), text.size());
}

void InternalTests::UartFlush()
{
    uart_flush(port);
}
```

Obrázok č. 29: : Podporné metódy

(Zdroj: vlastné spracovanie)

3.6 Vyhodnotenie testovania

Proces vyhodnotenia testovania nastane po tom čo príkaz TEST ukončí svoju činnosť, čiže keď všetky testy prebehnú. Ako som už uvádzal, každý test obsahuje výpisy, ktoré oznamujú výsledok procesu. Takže samotné vyhodnotenie je postavené na výpise týchto informačných logov, kde každý modul je pomenovaný a spoločne stým je aj výsledok testu. Teda je možné určiť, aký konkrétny modul uspel alebo zlyhal v teste.

Vo vyhodnocovaní testov je jediný rozdiel pri teste UARTu. Ako bolo spomínane, UART sa testuje vo viacerých operáciách v rámci fungovania celej aplikácie. Teda ak sa podarilo dopracovať k výpisom výsledkov testov modulov, tak UART sa považuje za funkčný a pripravený na fungovanie.

3.7 Prínosy

Táto implementácie prináša viacero prínosov, či už v rámci implementácie alebo samotnom fungovaní a procesov spojenými s testovaním.

V prvom rade, sa zameriam na opis prínosov v rámci implementácie. Hlavným prínosom z tejto časti, je to, že táto testovacia funkcionalita je obsiahnutá v samostatnom firmvéri, na rozdiel od pôvodnej implementácie. Tým, že je to samostatne implementované, tak sa zredukovala kódová pamäť produkčného firmvéru, kde to bolo zbytočné a neefektívne. Tak isto nadväznosť, ktorá bola medzi komunikačnými rozhraniami I2C v produkčnej verzii spôsobovala nefunkčnosť v istých stavoch a tým to samostatným riešením, sa podarilo odstrániť aj túto problematickú časť. V ďalšom kroku, sa mi podarilo vyriešiť aj problém s testovaním WiFi, kde bolo potrebné zmeniť vstupnú inicializáciu a dať tejto činnosti vyšší časový interval, kvôli požiadavkám ovládačov, ktoré neboli dovtedy korektne implementované.

Z hľadiska fungovania, na základe zistených nedostatkov funkcionality, bola implementovaná metóda na odstránenie pomocných testovacích súborov, čo primárne ušetrí využitú pamäť úložiska, keďže tieto súbory by sa tam nachádzali na trvalo a zaberali zbytočné miesto. Ďalšou výhodou tejto funkcionality, je to, že pri testovaní WiFi sa vytváral konfiguračný súbor, ktorý obsahoval dáta k pripojeniu sa k AP. Keďže

ten tam zostal uložený tak v prípade zákazníka sa mohlo udiať, že sa modul ESP32 snažil pripájať na AP na základe dát, ktoré mal uložené z testovacieho procesu.

Podstatným bodom je ešte vytvorenie nového príkazu SCAN. V praxi sa mohlo a aj v istých prípadoch udialo to, že test I2C vyhlasoval pri každom testovanom module chybu, no dôvod sa zistil až po istom čase. Bol taký, že testovací prípravok, ktorý ma v sebe zabudované I2C EEPROM pre testovanie, tie však boli poškodené, čiže výsledok testovania nemohol byť považovaný za relevantný. Z toho dôvodu, bolo potrebné zabezpečiť aj otestovanie korektnosti testovacieho prípravku, no a to príkaz SCAN umožňuje. Čiže je možné pred samotným testovaním overiť, či platforma, ktorá poskytuje prostredie pre testovanie korektne funguje.

Celkovým prínosom je zabezpečenie otestovania MCU ako výrobku a tým možnosť identifikácie chybových zariadení s konkrétnym pomenovaním toho, aký komponent je chybne osadený tým zjednodušiť prácu výrobným opravám, o potrebu identifikácie.

3.8 Ekonomické zhodnotenie

Vývoj tejto aplikácie bol realizovaný pomocou rozhraní pre vývoj, preklad kódu a nahrávanie firmvéru.

Pod pojmom rozhranie je myslené editor, ktorý bol použitý na písanie zdrojového kódu. Jedná sa Visual Studio Code, ktorý je voľne dostupný na používanie, čiže nespadá do skupiny rozhraní, na ktoré je nutné zakúpiť alebo predplatiť určitý typ licencie.

Samotný preklad kódu, alebo teda buildovanie bolo realizované pomocou CMake a predpripravených skriptov. Tieto náležitosti poskytuje tvorca a výrobca ESP32 bezplatne. Táto istá forma bezplatnosti využívania platforiem od ESPS32 platí aj pre proces nahrávania samotného firmvéru do MCU.

Na fungovanie tejto testovacej aplikácie je samozrejme potrebný MCU ESP32, kde sa daný firmvér dokáže nahrať a teda umožňuje výkon a chod tejto aplikácie. ESP32 bol použitý v dvoch typoch modulov a to ESP32-WROOM-32 a ESP32-WROVER-B, ktoré má spoločnosť Elcom vo vlastníctve.

Testovací proces je realizovaný pomocou testovacích výrobkov, ktoré spoločnosť Elcom už mala zaobstarané pred procesom návrhu a vývoja tejto aplikácie.

ZÁVER

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať testovaciu aplikáciu pre MCU ESP32 v spoločnosti Elcom. V prvých krokoch, riešenia tejto problematiky, bolo potrebné nadobudnúť podrobnejšie informácie ohľadom fungovania samotného ESP32 a možností, ktoré ponúka a tak isto korektne pochopiť jednotlivé hardvérové komponenty, ktoré sú predmetom testovania. Následne po nadobudnutí týchto znalostí bolo potrebné vyhodnotiť vstupnú analýzu momentálnej implementácie, identifikácie chybovostí a pripraviť si všetky potrené výstupy nato, aby bolo možné realizovať návrh riešenia, ktorý bude predstavovať najlepšie možnú realizáciu bez chýb a nedostatkov, ktoré sa podarilo odpozorovať. Tieto výstupy, slúžili v ďalšej kapitole ako vstup pre návrh, na základe ktorého, sa podarilo vyhotoviť návrh riešenia a následne pracovať na implementácii.

Implementácia prebehla do takého stavu, že momentálne sa výstup tejto práce reálne využíva vo výrobnom procese ESP32 v spoločnosti Elcom. Okrem odstránenie chybovosti a prínosu samostatnej aplikácie, ktorá je reálne použiteľná, boli pridané aj nové funkcionality, ktoré finálny výstup tejto práce posunuli na vyšší level.

Prínosy tejto práce sú na oboch stranách, samozrejme spoločnosť Elcom, ktorá obdržala produkt na základe jej požiadaviek, no ale aj prínosné pre moju osobu, oblasť IoT technológii je veľmi zaujímavá a rozmanitá a poskytuje obrovské možnosti využitia a počas práci na tejto problematike, som si vedomosti v tejto oblasti rozšírili a zdokonalil.

ZOZNAM POUŽITÝCH ZDROJOV

[1] What is internet of things (IoT)? *techtarget* [Online]. [cit. 2022-02-04]. Dostupné z: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

[2] What is IoT? *Oracle* [Online]. [cit. 2022-02-04]. Dostupné z: https://www.oracle.com/internet-of-things/what-is-iot/

[3] What is the Internet of Things (IoT)? *sap* [Online]. [cit. 2022-02-08]. Dostupné z: https://www.sap.com/insights/what-is-iot-internet-of-things.html

[4] 8 Examples of Internet of Things in Daily Life. *studiousguy* [Online]. [cit. 2022-02-01]. Dostupné z: https://studiousguy.com/examples-internet-of-things/

[5] ESP32. *espressif* [Online]. [cit. 2022-02-01]. Dostupné z: https://www.espressif.com/en/products/socs/esp32

[6] ESP32 Series of Modules. *espressif* [Online]. [cit. 2022-02-01]. Dostupné z: https://www.espressif.com/en/products/modules/esp32

[7] ESP32WROOM32 Datasheet. *espressif.com* [Online]. [cit. 2022-02-11]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[8] ESP32 Series Datasheet. *espressif* [Online]. [cit. 2022-02-11]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[9] ESP32WROVERB & ESP32WROVERIB Datasheet. *espressif* [Online]. [cit. 2022-02-11]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wrover-b_datasheet_en.pdf

[10] ESP32 I2C Communication: Set Pins, Multiple Bus Interfaces and Peripherals (Arduino IDE). *randomnerdtutorials* [Online]. [cit. 2022-02-11]. Dostupné z: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/

[11] Using the I2C Bus. *obot-electronics.* [Online] [cit. 2022-02-10]. Dostupné z: https://www.robot-electronics.co.uk/i2c-tutorial

[12] ESP32 Hardware Serial2 Example. *circuits4you.* [Online] [cit. 2022-02-11]. Dostupné z: https://circuits4you.com/2018/12/31/esp32-hardware-serial2-example/

[13] Universal Asynchronous Receiver/Transmitter (UART). *espressif* [Online]. [cit. 2022-02-01]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/uart.html

[14] Overview of ESP32 features. What do they practically mean? *exploreembedded* [Online]. [cit. 2022-02-01]. Dostupné z: https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F

[15] ESP32 Technical Reference Manual. *espressif* [Online]. [cit. 2022-02-01]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf#sysmem

[16] Support for External RAM. *espressif* [Online]. [cit. 2022-02-19]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/external-ram.html#external-ram-config-capability-allocator

[17] What is eMMC? (Embedded Multi-Media Card). *simms* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.simms.co.uk/insights/tech-talk/what-is-emmc-embedded-multi-media-card/

[18] Operating System - Overview. *tutorialspoint* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.tutorialspoint.com/operating_system/os_overview.htm

[19] What is an RTOS? *highintegritysystems* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.highintegritysystems.com/rtos/what-is-an-rtos/

[20] What is An RTOS? *freertos* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.freertos.org/about-RTOS.html.

[21] ESP-IDF FreeRTOS (SMP). *espressif* [Online]. [cit. 2022-02-19]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html.

[22] C++. *Britannica* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.britannica.com/technology/C-computer-language

[23] C Programming Language (C). *techopedia* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.techopedia.com/definition/24068/c-programming-language-c

[24] ESP32 Useful Wi-Fi Library Functions (Arduino IDE). *Randomnerdtutorials* [Online]. [cit. 2022-02-19]. Dostupné z: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

[25] Electrically Erasable Programmable Read-Only Memory (EEPROM). *Techopedia.* [Online]. [cit. 2022-02-19]. Dostupné z: https://www.techopedia.com/definition/2771/electrically-erasable-programmable-read-only-memory-eeprom

[26] Inter-Integrated Circuit (I2C). *docs.espressif* [Online]. [cit. 2022-02-19]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2c.html

[27] Ethernet. *docs.espressif* [Online]. [cit. 2022-02-19]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_eth.html

[28] What is TCP/IP and How Does it Work? *avast.* [Online[cit. 2022-03-08] Dostupné z: https://www.avast.com/c-what-is-tcp-ip

[29] What is a Transmission Control Protocol TCP/IP Model? *fortinet.* [Online] [cit. 2022-03-08] Dostupné z: https://www.fortinet.com/resources/cyberglossary/tcp-ip.

[30] Wi-Fi. *espressif.* [Online] [cit. 2022-03-08] Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html.

[31] IEEE 802.11x. *techopedia.* [Online] [cit. 2022-03-08] Dostupné z: https://www.techopedia.com/definition/508/ieee-80211x.

[32] O nás. *elcom.* [Online[cit. 2022-03-08] Dostupné z: https://www.elcom.eu/top/onas/.

[33] SWOT Analýza. *euroekonom* [Online]. [cit. 2022-03-08] Dostupné z: https://www.euroekonom.sk/manazment/strategicka-diagnostika/swot-analyza/

ZOZNAM OBRÁZKOV