FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Aktualizace zavaděče MCUboot pro platformu ESP32 a real-time operační systém Zephyr
Student:	Bc. David Horák
Vedoucí:	Ing. Tomáš Beneš
Studijní program:	Informatika
Obor / specializace:	Návrh a programování vestavných systémů
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2025/2026

Pokyny pro vypracování

Cílem práce je rozšířit podporu zavaděče MCUboot pro platformu ESP32 v real-time operačním systému Zephyr. Zephyr je "Linux Foundation Project", který v posledních letech získal trakci vývojové komunity a adoptuje koncepty z vývoje linuxového jádra. Zephyr je open-source real-time operační systém pro vestavné aplikace navržený tak, aby byl snadno použitelný, bezpečný, škálovatelný a podporuje širokou škálu architektur a hardwarových platforem.

1. Nastudujte a zdokumentujte funkční principy zavaděče MCUboot a operačního systému Zephyr.
2. Seznamte se s nástrojem MCUmgr, který slouží k aktualizaci firmware v operačním systému Zephyr.
3. Navrhněte a implementujte podporu pro aktualizace zavaděče MCUboot na platformě SoC ESP32-C3. Výsledná implementace musí podporovat secure boot a být schopná zotavit se z případných chyb během aktualizace (ztráta napájení, poškození přenášených dat, ...).
4. Během návrhu počítejte s případným budoucím rozšířením podpory o další SoC z rodiny ESP32.
5. Na závěr vzhledem k delikátní povaze této práce, otestujte výslednou implementaci pomocí testovacích scénářů, které budou obsahovat i simulaci chyb.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 27. února 2024 v Praze.

Diplomová práce

AKTUALIZACE ZAVADĚČE MCUBOOT PRO PLATFORMU ESP32 A REAL-TIME OPERAČNÍ SYSTÉM ZEPHYR

Bc. David Horák

Fakulta informačních technologií
Katedra číslicového návrhu
Vedoucí: Ing. Tomáš Beneš
9. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu Ing. Tomáši Benešovi za cenné rady a připomínky. Dále bych chtěl také poděkovat své rodině a přátelům za podporu během mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. května 2024

Abstrakt

Tato diplomová práce se zabývá přidáním podpory pro aktualizace zavaděče MCUboot v rámci operačního systému reálného času Zephyr pro platformu ESP32. Na začátku práce je představena platforma ESP32, zavaděč MCUboot a operační systém reálného času Zephyr. Pro vyřešení problému byl zavaděč MCUboot použit jako druhý aktualizovatelný zavaděč a pro platformu ESP32 byl napsán nový první zavaděč, který dovoluje bezpečné aktualizace zavaděče MCUboot. Aktualizaci zavaděče MCUboot provádí sám zavaděč MCUboot v kombinaci s jeho podporou pro více aktualizovatelných aplikací. Díky tomu je možné zavaděč aktualizovat stejným způsobem jako aplikaci. Součástí práce je vzorový projekt a test aktualizací, který slouží pro ověření funkčnosti.

Klíčová slova zavaděč, aktualizace zavaděče, aktualizace firmwaru, ESP32, MCUboot, RTOS, Zephyr RTOS

Abstract

This diploma thesis deals with adding support for updating MCUboot bootloader for ESP32 platform within real-time operating system Zephyr. In the begining of this thesis there is description of ESP32 platform, MCUboot bootloader and real-time operating system Zephyr. To solve the problem, MCUboot bootloader was used as second stage bootloader and new first stage bootloader was created for ESP32 that allows secure updates of MCUboot bootloader. Updating MCUboot bootloader is done by MCUboot itself. MCUboot supports updating multiple images, this was used to update both application and bootloader. The thesis includes sample project and test of firmware updates to verify its functionality.

Keywords bootloader, bootloader update, firmware update, ESP32, MCUboot, RTOS, Zephyr RTOS

Kapitola 1
Úvod

Vestavěné systémy začaly svou cestu jako jednoduché regulátory a řídicí jednotky v průmyslových zařízeních a spotřebičích. Příkladem mohou být mikrokontroléry používané v průmyslových strojích nebo v domácích spotřebičích. S postupem času se však rozšířily do širšího spektra aplikací, včetně lékařských přístrojů, automobilových systémů, mobilních zařízení, a dokonce i do zařízení nositelné elektroniky. Historicky byla tato zařízení naprogramována z továrny a nebylo možné je aktualizovat jinak, než v servisním středisku. Právě aktualizace softwaru zaznamenaly v poslední době velké změny. Dnes je běžné aktualizovat mobilní zařízení, automobil a nebo dokonce i ledničku.

Moderní vestavěné systémy jsou obvykle založeny na vyspělých mikroprocesorech a mikrokontrolérech, které umožňují rychlé zpracování dat a efektivní řízení zařízení. Tyto systémy často integrují senzory pro sběr dat z okolního prostředí, a také komunikační rozhraní pro propojení s dalšími zařízeními nebo s internetem. Díky této vysoké úrovni integrace a výkonu se vestavěné systémy staly základem pro internet věcí (IoT).

V současné době se stále častěji setkáváme s výkonnějšími procesory pro použití ve vestavných systémech. Dnes je možné setkat se s systémy na čipu s procesory o více jádrech a násobně větší velikostí paměti RAM, než bylo běžné například před 10 lety. S vyšším výkonem se otevírají možnosti pro řešení komplexnějších úloh. Dnešní moderní software pro komplexnější vestavěné systémy zpravidla staví na operačních systémech reálného času. Operační systémy reálného času poskytují více funkcionality a řeší problémy spojené se zpracováním více úloh najednou.

Společně s větším výpočetním výkonem dnešní systémy na čipu stále častěji nabízí bezdrátovou síťovou konektivitu. Součásti našich životů se v dnešní době stávají stále častěji IoT zařízení, ať už je to doma, na ulici, ve škole nebo v práci. Zařízení nám slibují ulehčení života, což ne vždy může být pravda. Takové zařízení je připojené k internetu a při nedostatečném zabezpečení může být zneužito. Již došlo k několika případům, kdy se útočníci dostali do takovýchto zařízení a zneužili je. Konkrétně došlo například ke zneužití více než tisíců CCTV kamer pro účely DDOS útoku na webové servery. Moderní systémy musí umožňovat aktualizaci software, aby bylo na zařízení možné aplikovat bezpečnostní záplaty. Velkou zranitelnost může představovat například několik let neaktualizovaný síťový router. [1] [2]

U moderních zařízení se síťovou konektivitou se aktualizace mohou distribuovat po síti. S tím se pojí další zranitelnosti. Útočník může podvrhnout soubor s aktualizací a zařízení tak dostane modifikovaný software, který může být použitý k nekalým účelům. Standardem se v dnešní době stal secure boot. Procesor podporující secure boot umožňuje spuštění pouze důvěryhodného kódu. Vydavatel softwaru aktualizaci podepíše klíčem uloženým na bezpečném místě. Zařízení je poté schopné spustit pouze kód podepsaný správným klíčem. Útočník může během přenosu aktualizace modifikovat její data. Zařízení ale kód nespustí, dokud není aktualizace znovu podepsaná správným klíčem.[2]

Důvodem pro aktualizace není jen bezpečnost. Aktualizace přinášejí novou funkcionalitu a je možné pomocí nich opravit chyby. Cílem je aby většina funkcionality zařízení byla aktualizovatelná. Během aktualizace není možné přímo přepisovat aktuálně běžící aplikaci. Během výpadku napájení by se zařízení mohlo dostat do stavu, kdy neobsahuje spustitelnou aplikaci. Z tohoto důvodu se software zařízení nejčastěji dělí na dvě části, zavaděč a aplikaci. Samotná aplikace neprovádí aktualizaci firmwaru, pouze poskytuje rozhraní pro nahrání nového softwaru do zařízení. O aktualizaci se typicky stará právě zavaděč, který na základě dat ve flash paměti provede aktualizaci. Zavaděč zároveň také často poskytuje funkcionalitu pro uvedení zařízení do funkčního stavu po neúspěšné aktualizaci. Umožňuje do zařízení nahrát software, pomocí USB, sériové komunikaci nebo jiným způsobem. Zavaděč je část softwaru, kterou obvykle není možné aktualizovat, a proto by měl poskytovat pouze minimum funkcionality. Zároveň je žádoucí aby využíval minimální množství zdrojů zařízení. Existují případy, kdy je třeba aby zavaděč poskytoval více funkcionality a byl komplexnější. Takovým příkladem může být například zavaděč, který dokáže spouštět aplikace nacházející se na vzdáleném síťovém úložišti. S komplexnějším softwarem se také pojí větší pravděpodobnost výskytu chyby. Pro takové případy se hierarchie zavaděčů dělí do více stupňů. První implementuje pouze minimum funkcionality a bývá neměnný. Druhý zavaděč poskytuje rozšířenou funkcionalitu a může být aktualizovatelný. [3]

1.1 Cíle práce

Ve firmě, kde autor pracuje, se dosud používaly procesory z řady nRF52 a nRF91 od výrobce Nordic Semiconductor. Nově se připravují produkty, které vyžadují Wi-Fi konektivitu. Právě kvůli zmíněné Wi-Fi konektivitě budou nová zařízení obsahovat procesory z rodiny ESP32. Stávající produkty staví na operačním systému Zephyr společně se zavaděčem MCUboot. Způsob, jakým jsou vyřešené aktualizace firmwaru, umožňuje kromě aktualizace aplikace také aktualizace zavaděče.

Cílem této práce je přidání podpory aktualizace zavaděče MCUboot v rámci operačního systému Zephyr také pro platformu ESP32, která se bude využívat na budoucích produktech. Konkrétně jde o systémy na čipu ESP32-C3 a ESP32-S3. Výsledné řešení musí zajišťovat podporu pro secure boot a aktualizace musejí být odolné vůči chybám, které se mohou během aktualizace vyskytnou. Zároveň by řešení mělo být navržené tak, aby bylo snadno rozšiřitelné o další procesory z rodiny ESP32. Aktualizaci zavaděče by mělo jít provést stejně jako aktualizace aplikace, pomocí nástroje MCUmgr.

Kapitola 2
Platforma ESP32

ESP32 je série levných mikroprocesorů s bohatou síťovou konektivitou od čínského výrobce Espressif Systems. Čip ESP32 byl představen jako nástupce čipu ESP8266. Starší ESP8266 bylo určeno především jako externí síťový koprocesor s Wi-Fi konektivitou. Pro aplikaci bylo nutné mít další mikroprocesor. V roce 2014 firma Espressif vydala SDK, které dovolilo naprogramovat ESP8266 přímo a odstranila se tak potřeba mít separátní procesor pro aplikaci. Stále zde však byla limitace v počtu GPIO pinů a nedostatku periferií. [4]

Tyto nedostatky se firma snaží řešit a v roce 2016 představuje SoC ESP32, které přináší rychlejší dvou-jádrový procesor, více paměti RAM, téměř dvojnásobný počet GPIO pinů a modernější bezdrátovou konektivitu. Nově je zde podpora pro Bluetooth LE ve verzi 4.2. [4]

V dnešní době zahrnuje rodina několik variant ESP32. Mikroprocesory využívají jádra s architekturou Xtensa a nebo RISC-V. Síťová konektivita zahrnuje především technologii Wi-Fi a Bluetooth, včetně Bluetooth LE. Nově jsou v portfóliu také SoC s konektivitou pomocí IEEE 802.15.4, jde o čipy ESP32-H2 a ESP32-C6. Vzhledem ke své konektivitě se tyto procesory často používají v IoT aplikacích. [4]

Oficiální vývojová platforma je ESP-IDF, která je založená na operačním systému reálného času FreeRTOS. Není však třeba využívat pouze softwarovou platformu ESP-IDF, kromě ni existuje podpora také v operačním systému Zephyr a Apache NuttX. [4]

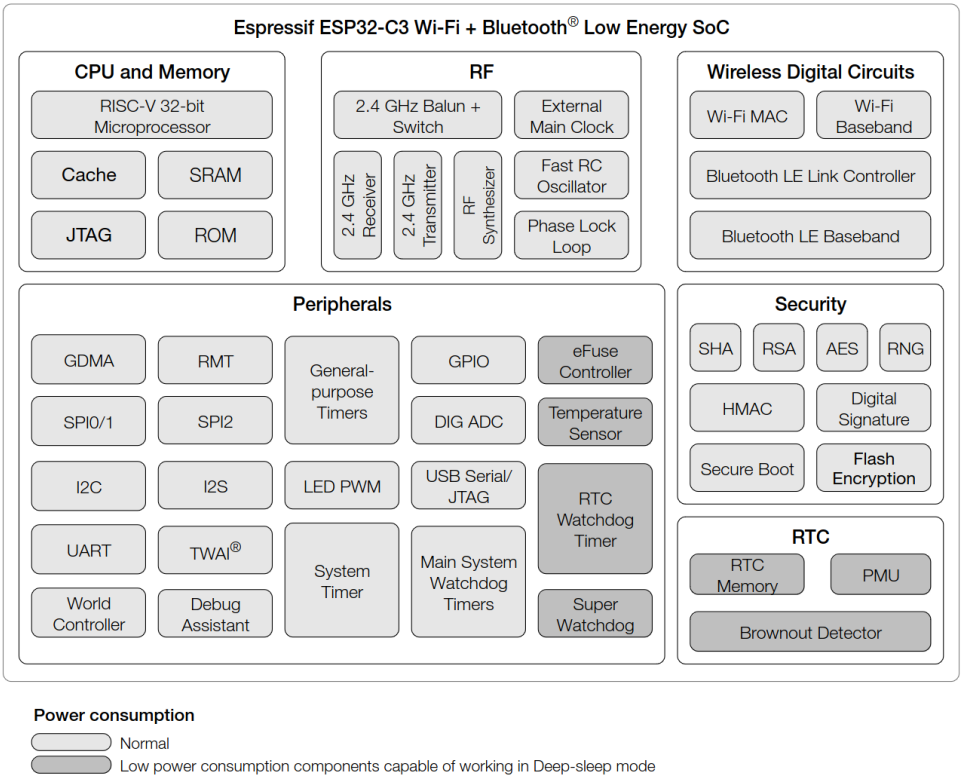
2.1 SoC ESP32-C3

V rámci této práce autor používá především čip ESP32-C3. Výsledné řešení bylo v průběhu portované také na čip ESP32-S3. V počátku práce byl však čip ESP32-C3 ten, pro který bylo řešení navrhováno. většina věcí je však na všech čipech ESP32 řešena stejně, proto je zde uveden jen popis čipu ESP32-C3.[5] [6] [7] [8]

ESP32-C3 je čip s 32-bitovým jedno-jádrovým procesorem s architekturou RISC-V, který může být taktován maximálně na 160 MHz. Procesor je doplněn pamětí RAM o velikosti 400 KB. Z celkové velikosti paměti je 16 KB možné použít jako cache paměť pro přístup k instrukcím. Dále je zde 8 KB RTC RAM paměti. Tato paměť si udrží svá data i v případě, že je čip uveden do deep-sleep režimu. [7]

Na čipu se také nachází 384 KB paměti ROM, do které je z výroby „vypálen“ zavaděč a zdrojový kód, který je přístupný z aplikace. Zavaděč v ROM paměti může být označován také jako nultý zavaděč. Tuto paměť není možné žádným způsobem uživatelsky modifikovat. K čipu je možné připojit externí flash paměť pomocí rozhraní SPI. Flash paměť je možné pomocí MMU namapovat přímo do adresního prostoru. Díky tomu lze instrukce a data dostupná pouze pro čtení linkovat přímo do externí flash paměti. [5] [9]

Bezdrátovou konektivitu zajišťuje technologie Wi-Fi a Bluetooth. ESP32-C3 podporuje Wi-Fi standardu IEEE 802.11b/g/n a Bluetooth LE ve verzi 5 včetně rozšíření Extended Advertising a Long Range. Čip obsahuje periferie rozhraní SPI, UART, I2C, I2C a mnoho dalších. [7]



Obrázek 2.1 Blokový diagram SoC ESP32-C3 [7]

2.2 Paměť

Mapa adresního prostoru čipu ESP32-C3 je zobrazena na obrázku 2.2. Na čipech ESP32 architektury Xtensa je adresní prostor řešený úplně stejně, liší se jen v hodnotách adres. Přístup do pamětí (externí flash, ROM, RAM) je realizován pomocí dvou sběrnic, instrukční a datové. Pomocí instrukční sběrnice lze přistupovat také k datům, ale adresa musí být zarovnaná na 4 byty, oproti tomu datová sběrnice umožňuje přistupovat k datům po 1 bytu. [5] [6]

Na čipu jsou integrovány následující typy pamětí:

ROM

Jde paměť dostupnou pouze pro čtení, do které jsou z výroby zapsáná data. Nachází se zde ROM zavaděč společně s daty a instrukcemi systémového software (část kódu pro přístup k rádiové periferii, funkce pro čtení flash paměti, část standardní knihovny jazyka C, ...). Na čipu se nachází 384 KB paměti tohoto typu. Prvních 256 KB je dostupných pouze skrz instrukční sběrnici a zbylých 128 KB pomocí instrukční a datové sběrnice.

RAM

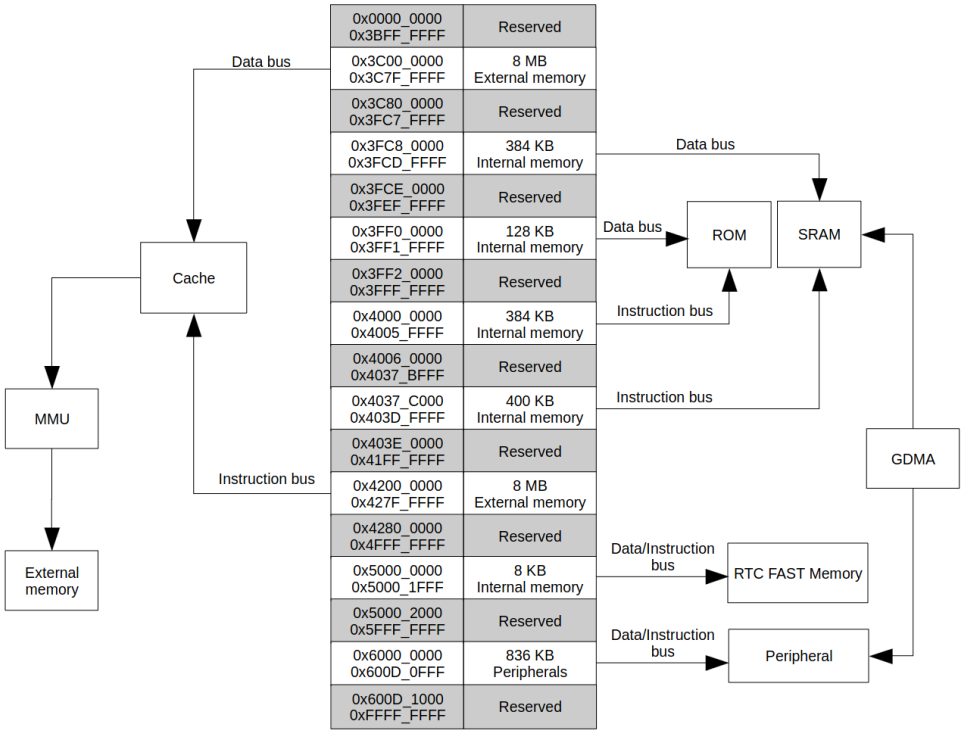
Ne všechny části paměti RAM jsou dostupné pomocí instrukční a datové sběrnice zároveň. Prvních 16 KB paměti, která jde zároveň použita jako instrukční cache, je dostupná pouze pomocí instrukční sběrnice. Ke zbylým 384 KB je možné přistupovat pomocí obou sběrnic.

RTC RAM

Jde o standardní RAM paměť, která je persistentní i v deep-sleep režimu. Na čipu je k dispozici 8 KB této paměti. Přistupovat k ní lze pomocí jedné sběrnice společné pro data i instrukce. [5]

K čipu je také možné pomocí SPI připojit externí flash paměť. Paměť lze díky MMU namapovat přímo do adresního prostoru. Až 8 MB lze namapovat jako data, dalších 8 MB jako instrukce. Flash pamět se mapuje po blocích velkých 64 KB. Aplikace se standardně nenachází pouze v paměti RAM, ale většina instrukcí a data pro čtení jsou uložena v externí flash paměti. Během linkování aplikace je symbolům nacházejícím se ve flash paměti přiražena odpovídající virtuální adresa. Zavaděč během zavádění aplikace namapuje části flash paměti do adresního prostoru. Z důvodu mapování je nutné, aby byl obraz aplikace uložen v paměti na adrese, která je násobkem 64 KB (velikost stránky MMU). [5]

Pokud se v této práci vyskytne výraz IRAM, DRAM, je tím myšlena paměť RAM, ke které se přistupuje pomocí instrukční, respektive datové sběrnice. Obdobně budu označovat také externí flash paměť pomocí zkratky IROM / DROM.



Obrázek 2.2 Mapa adresního prostoru SoC ESP32-C3 [5]

2.3 eFuse

Všechny SoC platformy ESP32 obsahují speciální jednorázově programovatelnou paměť eFuse (elektronické pojistky). Paměť je bitově orientovaná. Při práci s eFuse je třeba velké opatrnosti. Zápis je jednorázová operace. Jakmile je bit nastaven na logickou jedničku, neexistuje žádný způsob jak bit resetovat zpět do nuly.

Dle technické dokumentace obsahuje SoC ESP32-C3 4096 jednorázově programovatelných bitů, avšak dále se v dokumentu uvádí, že eFuse jsou rozděleny do 11 bloků po 256 bitech. To odpovídá pouze 2816 bitům. [5]

Blok 0-2
Bloky jsou vyhrazené pro systémové účely. Nastavením některých bitů lze například aktivovat secure boot, vypnout rozhraní JTAG, zakázat nahrávání firmwaru pomocí UART nebo nastavit ochranu před čtením/zápisem určitým eFuse blokům, Také se zde nachází výrobní data specifická pro konkrétní čip, zejména MAC adresa a kalibrace ADC.

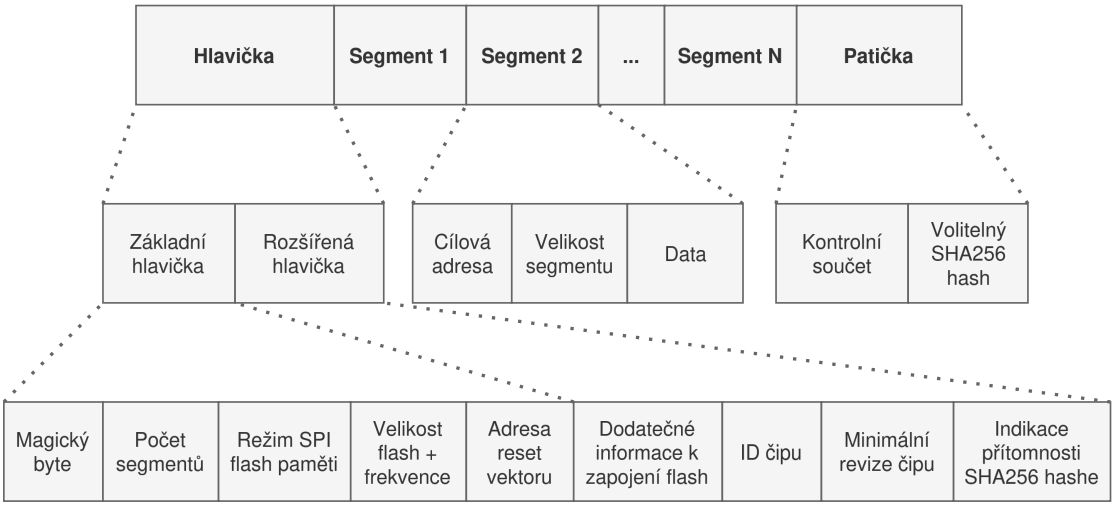
Blok 3
Je vyhrazen pro uživatelská data.

Blok 4-9
Bloky jsou vyhrazené pro ukládání klíčů. Mohou zde být uloženy klíče pro secure boot a pro šifrování externí flash paměti. Výjimku tvoří blok 9, do kterého není možné uložit klíč pro šifrování paměti, důvodem je hardwarová chyba. Pokud některý z bloků není obsazen klíčem, je možné jej využít pro uživatelská data.

Blok 10
Dle dokumentace je opět vyhrazený pro systémové využití. V tomto bloku se ovšem nenachází žádná data a nikde není zmíněno jeho využití, nejspíš ho lze také využít pro uživatelská data. [9]

2.4 Formát obrazu firmware

Aby byl ROM zavaděč schopný zavést firmware, je třeba dodržet specifický formát. Tento formát je společný pro všechny čipy platformy ESP32. Struktura firmware je vyobrazena na obrázku 2.3.



Obrázek 2.3 Formát obrazu firmware pro ESP32 [9]

Hlavička firmware je rozdělena na dvě hlavičky, základní a rozšířenou. V hlavičce jsou obsaženy následující údaje:

Magický byte
Podle něj je možné detekovat, zda se v paměti nachází firmware v daném formátu.

Počet segmentů
Udává kolik je v obraze přítomných segmentů s daty.

Režim SPI flash paměti
Udává v jakém SPI režimu je externí paměť zapojena (QIO, QOUT, DIO, DOUT).

Velikost flash paměti a její frekvence
Jeden byte. Horní 4 bity udávají velikost paměti, dolní 4 bity její frekvenci.

Adresa reset vektoru
Adresa v paměti, na kterou se skočí po úspěšném zavedení firmware.

Dodatečné informace k zapojení flash
Určuje, jak je zapojena externí flash pomocí SPI k čipu, pokud je tomu jinak oproti výchozímu stavu.

ID čipu
Určuje, pro jaký čip je firmware určený.

Minimální revize čipu
Minimální revize, se kterou je firmware kompatibilní.

Indikace přítomnosti hashe
Indikuje zda se v patičce nachází hash obrazu firmwaru [9]

Po hlavičce následují samotné datové segmenty. Podle cílové adresy segmentu se zavaděč rozhodne zda daný segment nahraje do paměti RAM, či namapuje do adresního prostoru. Pokud cílová adresa odpovídá paměti IRAM či DRAM je segment nahrán. V případě, že adresa odpovídá IROM nebo DROM, segment je namapován do adresního prostoru. V patičce obrazu se vždy nachází kontrolní součet a volitelně také, hash obrazu. [9]

Již byla zmíněna přítomnost dvou sběrnic pro přístup do paměti. S tím také souvisí formát firmware. Část firmware je do paměti nahrávána pomocí instrukční sběrnice a část pomocí datové. RAM paměť je tedy vždy rozdělena na minimálně dva segmenty. Stejně je tomu v případě mapování, zvlášť jsou namapovány segmenty s instrukcemi a segmenty obsahující data, opět minimálně dva segmenty. Segmenty, které se do adresního prostoru mapují musí navíc splňovat podmínku zarovnání na velikost stránky, o to se stará nástroj `elf2image`, pomocí něj se ze souboru ve formátu ELF vytvoří binární soubor v právě zmiňovaném formátu. [5] [9]

2.5 Secure boot

Secure boot je bezpečnostní funkce, která pomáhá chránit zařízení před spuštěním neoprávněného kódu. Pokud je secure boot aktivován, zařízení je schopné spustit pouze podepsaný firmware z důvěryhodného zdroje. Původní čip ESP32 měl implementovaný secure boot na bázi symetrické šifry AES, jeho použití již není ze strany výrobce doporučené. Modernější čipy a novější revize ESP32 používají secure boot v2 na bázi asymetrické šifry RSA. Soukromý klíč slouží k podepsání firmwaru, veřejným klíčem je možné podpis ověřit. [9]

Aby mělo smysl využívat secure boot, je nutné ověřovat každou část software, kterou může procesor spustit. V případě použití ESP-IDF SDK jde o jakýkoliv aplikační firmware a zavaděč. Zavaděč v paměti ROM není nijak podepisován a není to potřeba, protože se nachází v interní ROM paměti a není možné ho žádným způsobem přepsat. [2] [9]

2.5.1 Formát podepsaného firmware

Podepsaný firmware se od nepodepsaného téměř neliší, jen je na jeho konec přidán blok obsahující podpis. Struktura podpisu je popsána v tabulce 2.1. Hodnoty `R` a `N` jsou odvozeny z modulu, respektive exponentu a jsou používány pro hardwarově akcelerované násobení v montgomerygo doméně. [9]

Tabulka 2.1 Struktura podpisového bloku [9]

offset	velikost (bytů)	popis
0	1	Magický byte
1	1	Verze secure boot
2	2	Výplň
4	32	SHA-256 hash obrazu firmware (bez tohoto bloku)
36	384	Modul používaný pro ověření podpisu
420	4	Exponent používaný pro ověření podpisu
424	384	Přepočítaná hodnota R pro násobení v montgomeryho doméně
808	4	Přepočítaná hodnota M pro násobení v montgomeryho doméně
812	384	Výsledek podpisu firmwaru pomocí RSA
1196	4	CRC32 předchozích 1196 bytů
1200	16	Doplnění do 1216 bytů

Součástí podpisového bloku je také veřejný klíč, který slouží k verifikaci firmware. To znamená, že je možné ověřit firmware podepsaný jakýmkoliv privátním klíčem. Veřejný klíč se vždy nejprve ověřuje. Hash veřejného klíče je uložen v eFuse. Do eFuse je možné uložit až tři klíče. Před ověřením firmware se vždy zkontroluje zda se veřejný klíč v podpisovém bloku shoduje s důvěryhodným klíčem uloženým v eFuse. V případě, že se klíče neshodují není firmware důvěryhodný a zavádění končí chybou. Firmware je tedy důvěryhodný pouze pokud je podepsán důvěryhodným klíčem a zároveň je úspěšně verifikován jeho podpis. [9]

2.5.2 Secure boot a MMU

Při použití secure boot v2 se doporučuje obraz firmware doplnit nulami tak, aby jeho velikost byla dělitelná velikostí stránky MMU. Standardně je stránka velká 64 KB. Podpis díky tomu bude zarovnaný na velikost stránky. Důvodem je, že zavaděč během zpracovávání podpisu mapuje flash paměť do adresního prostoru. Při nezarovnání podpisu by se v adresním prostoru mohl objevit i nedůvěryhodný kód, což představuje bezpečnostní riziko. Doplnění je do obrazu firmware možné přidat během vytváření obrazu pomocí nástroje `elf2image`, stačí použít parametr `--secure-pad-v2`. [9]

2.5.3 Postup při ověření firmware

Postup během ověřování firmware je následující:

1. Po startu čipu je spuštěn zavaděč nacházející se v paměti ROM.
2. Pokud není secure boot aktivován, verifikace je přeskočena. V opačném případě se pokračuje následujícími kroky.
3. Zavaděč ověří platnost podpisového bloku, to zahrnuje ověření magické bytu a ověření kontrolního součtu.
4. Následně je spočítán hash firmware a porovnán s hash uloženým v podpisovém bloku.

5. Jako další krok dojde k ověření veřejného klíče, spočítá se hash veřejného klíče, který se nachází v podpisovém bloku, a následně se porovná s klíči uloženými v eFuse.
6. Ověří se podpis firmware.
7. Pokud předchozí kroky proběhly úspěšně je spuštěn požadovaný firmware. [9]

2.5.4 Aktivace secure boot

Pro aktivaci secure boot je nutné do paměti eFuse vypálit bit `SECURE_BOOT_EN` a hash veřejného klíče. Součástí vývojářského balíku jsou nástroje, které usnadňují generování klíčů, jejich zápis do efuse a podepisování firmwaru. [9]

Výpis kódu 2.1 Vygenerování klíče pomocí espsecure.py

```
$ espsecure.py generate_signing_key key.pem --scheme rsa3072
```

Výpis kódu 2.2 Vygenerování klíče pomocí openssl

```
$ openssl genrsa -out key.pem 3072
```

Výpis kódu 2.3 Zápis klíčů do efuse

```
$ espefuse.py burn_key_digest \
            BLOCK_KEY0 key_0.pem  SECURE_BOOT_DIGEST0  \
            BLOCK_KEY1 key_1.pem  SECURE_BOOT_DIGEST1  \
            BLOCK_KEY2 key_2.pem  SECURE_BOOT_DIGEST2
```

Příkaz `burn_key_digest` zároveň automaticky nastaví odpovídajícím blokům ochranu proti zápisu. Jako parametr příkazu je možné použít veřejný nebo privátní. Z privátní klíče se vygeneruje veřejný, který je následně zapsán do eFuse. [9]

Výpis kódu 2.4 Aktivace secure boot

```
$ espefuse.py burn_efuse SECURE_BOOT_EN 1
```

Po aktivaci secure boot je možné spustit pouze firmware podepsaný odpovídajícím privátním klíčem. Příkaz `espsecure.py sign_data` analyzuje firmware a na jeho konec zapíše blok obsahující podpis. [9]

Výpis kódu 2.5 Podepsání firmware

```
$ espsecure.py sign_data \
            --version 2 \
            --pub-key key.pub \
            --output firmware.signed.bin \
            firmware.bin
```

2.5.5 Šifrování flash paměti

Secure boot lze používat i bez šifrování flash paměti. Pokud však útočník získá fyzický přístup k zařízeni, může po ověření podpisu firmwaru a před jeho spuštěním změnit obsah flash paměti a zavaděč tak spustí nedůvěryhodný firmware. Tento problém řeší šifrování flash paměti, veškerá přenášená data po SPI mezi procesorem a paměťovým čipem jsou šifrovány pomocí AES. Klíč je opět uložen veFuse a tentokrát je chráněn proti čtení i zápisu, stále ale zůstává přístupný hardwarové jednotce, která se stará o šifrování dat. Společně se secure boot je silně doporučeno zapnutí šifrování externí flash paměti. [2] [9]

2.6 Podpůrné nástroje

Pro práci s čipy platformy ESP32 je distribuován balíček `esptool.py`, který obsahuje nástroje pro vytváření obrazu firmwaru, podepisování firmwaru a komunikaci s ROM zavaděčem. Právě prostřednictvím ROM zavaděče je možné číst a zapisovat data do externí flash paměti. Také je pomocí něj možné číst a zapisovat do eFuse. Nástroje jsou schopné komunikovat s čipem pouze pokud je uveden do download režimu. Na vývojové desce se čip do download režimu restartuje automaticky při použití nástroje `esptool.py`. [9]

Balíček nástrojů `esptool.py` je distribuován jako Python balíček. Instalaci je možné provést pomocí správce balíčků `pip`.

Výpis kódu 2.6 Instalace esptool.py

```
$ pip install esptool.py
```

2.6.1 Vytvoření obrazu firmware

Příkaz `elf2image` slouží k vytvoření obrazu firmwaru. Nástroj zkonvertuje sestavený firmware ve formátu ELF do binárního souboru ve formátu kompatibilním s ROM zavaděčem. [9]

Výpis kódu 2.7 Vytvoření obrazu firmwaru

```
$ esptool.py --chip esp32c3 elf2image my_app.elf
```

2.6.2 Nahrávání firmware

Pomocí `esptool.py` lze rovněž číst a zapisovat do externí flash paměti. Použití je především pro zápis firmwaru do flash paměti. [9]

Výpis kódu 2.8 Nahrání firmware pomocí esptool.py přes rozhraní UART

```
$ esptool.py --chip auto write_flash 0x10000 filename.bin
```

Kromě nahrávaní firmwaru prostřednictvím nástroje `esptool.py` a tedy skrze sériový port je možné firmware nahrát také pomocí rozhraní JTAG. K tomu je nutné mít stažené OpenOCD. [9]

Výpis kódu 2.9 Nahrání firmware pomocí OpenOCD přes rozhraní JTAG

```
$ openocd -f board/esp32c3-builtin.cfg \
          -c "program_esp filename.bin 0x10000 verify exit"
```

2.6.3 Čtení a zápis eFuse

Součástí balíku `esptool.py` je také nástroj `espefuse.py`, který umožňuje čtení a zápis dat do paměti eFuse. Následující příkaz vypálí bity 15, 16, 17, 18, 19, 20 v bloku 2 na logickou 1.

Výpis kódu 2.10 Vypálení bitů paměti eFuse

```
$ espefuse.py burn_bit BLOCK2 15 16 17 18 19 20
```

Data je množné vypálit také na základě obsahu binárního souboru.

Výpis kódu 2.11 Vypálení eFuse na základě obsahu binárního souboru

```
$ espefuse.py burn_block_data BLOCK3 efuse_data.bin
```

Pro zobrazení aktuálního stavu eFuse paměti slouží příkaz `espefuse.py summary`.

Výpis kódu 2.12 Zobrazení obsahu eFuse

```
$ espefuse.py summary
```

Kapitola 3
Zavaděč MCUboot

MCUboot je bezpečný zavaděč pro 32-bitové mikroprocesory. MCUboot nezávisí na žádném konkrétním operačním systému nebo hardwaru. Většina funkcionality je systémově nezávislá, systémově závislá funkcionalita (čtení flash, způsob zavádění firmware) je vždy implementována pro specifický operační systém, architekturu nebo SoC. Zavaděč umožňuje aktualizace firmwaru, recovery po sériové lince a je odolný vůči chybám během aktualizace. [10]

Zavaděč podporuje secure boot. MCUboot neimplementuje vlastní kryptografickou knihovnu, ale spoléhá se na Mbed TLS, případně TinyCrypt. Pokud není žádoucí použití těchto knihoven, je možné implementovat své vlastní kryptografické funkce. Na většině moderních SoC jsou hardwarově implementované kryptografické akcelerátory. MCUboot umožňuje využití těchto akcelerátorů, uživatel může implementovat kryptografické funkce využívající hardwarové akcelerace. Takto používá MCUboot na svých produktech firma Nordic Semiconductor, kde jsou kryptografické funkce implementované s pomocí hardwarového akcelerátoru ARM CryptoCell. Díky tomu je ověřování firmware rychlejší a v zavaděči nemusí být zakompilovány externí knihovny, což ušetří paměť RAM i flash. [10] [11]

Projekt MCUboot se skládá ze dvou hlavních částí, knihovny bootutil a samotné aplikace zavaděče. Knihovna bootutil implementuje většinu funkcionality zavaděče, včetně secure boot, podpory pro aktualizace firmware a recovery. Tato část je systémově nezávislá. Samotná aplikace zavaděče využívá zmíněnou knihovnu a implementuje poslední krok nutný k nastartování aplikace, zavádění a skok do aplikace. Tato část je závislá na architektuře systému, na různých architekturách má firmware jiný formát a tudíž je postup jiný. [10] [12]

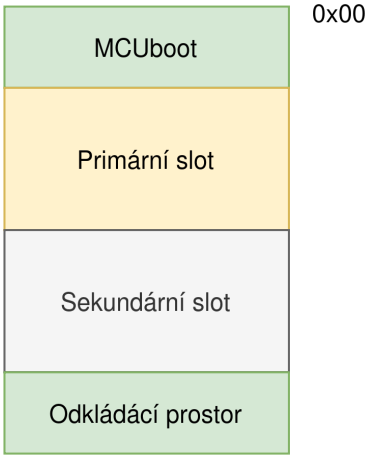
3.1 Rozdělení flash paměti

Z pohledu zavaděče MCUboot je flash paměť rozdělena do několika segmentů. V této práci jsou segmenty označovány jako sloty. Každá aplikace má ve flash paměti vyhrazené dva sloty, primární a sekundární. Standardně zavaděč zavádí aplikaci z primárního slotu. Aplikace je sestavená pro běh právě z tohoto primárního slotu. Sekundární slot slouží jako místo pro umístění nové verze. Rozdělení flash paměti je definované v hlavičkovém souboru `sysflash.h` pomocí maker. [10]

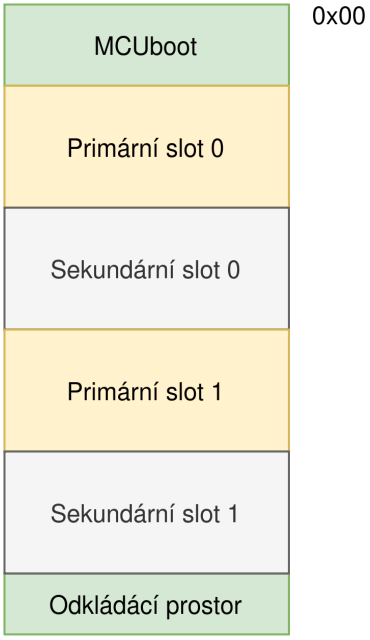
Při spuštěni zavaděč kontroluje, zda se v sekundárním slotu nenachází připravená aktualizace. Pokud je k dispozici aktualizace, zavaděč vymění obsah primárního a sekundárního slotu. V závislosti na konfiguraci k tomu může využít odkládací prostor. Výjimkou jsou speciální režimy aktualizace Direct-XIP a RAM-load, kdy může být aplikace zavedena z obou slotů. [10]

V dnešní době stále více SoC obsahuje více-jádrové procesory, které umožňuji AMP výpočet. Společně s AMP je také nutné distribuovat rozdílné aplikace pro jednotlivá jádra. Zavaděč MCUboot podporuje aktualizace více aplikací. Každá aplikace má svůj vlastní primární a sekundární

slot. Zavádění všech aplikací již není starost zavaděče MCUboot. Zavaděč provádí pouze zavedení hlavní aplikace, ta má následně na starosti zavedení ostatních aplikací pro další jádra. MCUboot také dokáže vyhodnotit závislosti mezi jednotlivými aplikacemi. [10]. Na obrázku 3.1 je zobrazeno možné rozdělení flash paměti pro jednu aktualizovatelnou aplikaci, na obrázku 3.2 rozdělení pro dvě aplikace. [10]



Obrázek 3.1 Příklad rozdělení flash paměti pro jednu aplikaci



Obrázek 3.2 Příklad rozdělení flash paměti pro dvě aplikace

3.2 Aktualizace firmware

Pokud je zařízení zapnuto, je spuštěn zavaděč MCUboot. Za normálních okolností je v primárním slotu nejnovější aplikace, a není tedy potřeba provádět výměnu slotů. Během aktualizace aplikace je do sekundárního slotu nahrána nová verze a zavaděč má za úkol provést výměnu slotů. Po spuštění se vždy vyhodnotí jakou operaci má provést pomocí aktuálního stavu výměny. [10]

Aktualizace může být dvou-krokový proces. Jako první krok MCUboot provede testovací výměnu slotů tím, že nastaví stav výměny na testovací a spustí novou aplikaci. Aplikace se následně může sama otestovat a poté nastaví stav výměny jako permanentní. Pokud by stav zůstal nastavený na testovací, při dalším spuštění zavaděč provede reverzní výměnu a vrátí tak zpět původní verzi aplikace. Již během nahrávání nové aplikace do sekundárního slotu má uživatel možnost nastavit stav výměny. Pokud uživatel nastaví permanentní stav, aplikace již nemusí nic potvrzovat, aktualizace zůstane permanentní. [10]

Stav výměny je uložen společně s dalšími metadaty v posledním sektoru slotu. Stav výměny slotů může nabývat následujících stavů:

Prázdný stav
Standardní stav, pokud není prováděná aktualizace. Výměna slotů není provedena.

Testovací stav
Vymění primární a sekundární slot. V případě že není stav nastaven na permanentní je při dalším spuštění stav nastaven na reverzní.

Permanentní stav
Provede se permanentní výměna slotů.

Reverzní stav
Předchozí testovací výměna neproběhla úspěšně. Zavaděč vymění obsah slotů zpět do původního stavu.

Chybový stav
Výměna se nezdařila, protože obsah sekundární slotu není validní.

Stav panika
Během výměny došlo k chybě, ze které se nelze zotavit. [10]

3.2.1 Způsob výměny slotů

MCUboot podporuje několik způsobů, jakým provádí aktualizaci. Během každého pracuje s primárním a sekundárním slotem. Způsoby aktualizace jsou následující:

Výměna pomocí odkládacího prostoru
Výměna pomocí tohoto způsobu využívá vyhrazený odkládací prostor ve flash paměti. Velikost odkládacího prostoru musí být větší nebo stejná jako velikost jedné stránky. Opotřebení flash paměti závisí na velikosti odkládací prostoru. Použití většího prostoru snižuje opotřebení flash paměti.

Výměna bez odkládacího prostoru
Výměna je také možná bez použití odkládacího prostoru. Postupně jsou sektory posouvány dokud nejsou obsahy slotů vyměněny. Algoritmus funguje následovně:

1. Všechny sektory v primárním slotu posuň o jeden výše. Nastav N=0.
2. Zkopíruj N-tý sektor sekundárního slotu do N-tého sektoru primárního slotu.
3. Zkopíruj N+1. sektor primárního slotu do N-tého sektoru sekundárního slotu.

4. Opakuj kroky 2 a 3 pro všechny sektory.

Výhoda tohoto způsobu je úspora flash paměti, bohužel za cenu většího opotřebení flash paměti.

Direct-XIP
V tomto režimu nedochází k výměně obsahu slotů, aplikace může být zavedena také přímo ze sekundárního slotu. Zavaděč provede inspekci primárního a sekundární slotu a na základě verze se rozhodne, kterou aplikaci zavede. Zaváděna je aplikace, která má vyšší verzi. V tomto režimu je také umožněno testovat aktualizaci aplikace. Místo provedení reverzní výměny je aplikace ze slotu odstraněna. Výhodou tohoto režimu je rychlejší průběh aktualizace a nižší opotřebení flash paměti. Velkou nevýhodou je potřeba sestavovat dvě verze aplikace, jednu pro primární slot a druhou pro sekundární.

Zavedení do RAM
Funguje téměř stejně jako režim Direct-XIP, rozdíl je v tom, že aplikace běží čistě v paměti RAM. [10]

3.2.2 Metadata

Na konci slotů je vyhrazené místo pro metadata. Pomocí nich zavaděč pozná aktuální stav a jakou operaci má během svého běhu provést. Metadata jsou uložena v obou slotech, primárním i sekundárním, a obsahují následující položky:

Magic
Slovo určující přítomnost metadat.

Swap status
Během výměny slotů provádí zavaděč různé operace se sektory. Zde je uložena operace, jaká se zrovna provádí. Pokud by došlo k výpadku napájení během aktualizace, zavaděč je díky těmto informacím schopný pokračovat ve výměně slotů.

Swap size
Počet sektorů, které je nutné vyměnit. Zpravidla se rovná velikosti větší aplikace v primárním nebo sekundárním slotu.

Swap info
Informace o tom jaká operace výměny slotů je právě prováděna (testovací, permanentní, reverzní).

Copy done
Hodnota indikující, že se aplikace nachází v cílovém slotu.

Image OK
Uživatelsky zapsaná hodnota, indikující, že je aplikace připravena k aktualizaci. [10]

MCUboot se podle informací v metadatech rozhodne jakou operaci provede. Zda-li došlo v předešlém běhu k přerušení aktualizace zavaděč pozná pomocí položek `swap info` a `swap status`. Následně pokračuje od bodu kde byla operace přerušena. [10]

Během nové operace výměny slotů (například po nahrání aktualizace do sekundárního slotu) nejsou položky `swap info` a `swap status` nastaveny. Zavaděč zjistí aktuální stav metadat primárního i sekundárního slotu a na základě položek `magic`, `image ok` a `copy done` se rozhodne jakou operaci provede. V tabulkách níže je uvedeno jak dojde k vyhodnocení výsledné operace. [10]

Tabulka 3.1 MCUboot - vyhodnocení operace I [10]

	primární slot	**sekundární slot**
magic	nezáleží	v pořádku
image ok	nezáleží	nenastaven
copy done	nezáleží	nezáleží
výsledná operace	**testovací výměna**	

Tabulka 3.2 MCUboot - vyhodnocení operace II [10]

	primární slot	**sekundární slot**
magic	nezáleží	v pořádku
image ok	nezáleží	nastaven
copy done	nezáleží	nezáleží
výsledná operace	**permanentní výměna**	

Tabulka 3.3 MCUboot - vyhodnocení operace III [10]

	primární slot	**sekundární slot**
magic	v pořádku	nenastaven
image ok	nenastaven	nezáleží
copy done	nastaven	nezáleží
výsledná operace	**reverzní výměna**	

3.2.3 Kroky aktualizace aplikace

Níže jsou uvedeny kroky během aktualizace aplikace od spuštění zavaděče MCUboot až po zavedení aplikace:

1. Zkontroluj stav výměny slotů, byla předchozí operace přerušena?
   - Ano: Dokonči výměnu slotů.
   - Ne: pokračuj na krok 2.
2. Zkontroluj metadata, je vyžadována výměna slotů?
   - Ano: Proveď výměnu slotů, pokud je aplikace v sekundárním slotu úspěšně ověřena, jinak aplikaci odstraň a pokračuj na krok 3.
   - Ne: Pokračuj na krok 3.
3. Ověř integritu, případně důvěryhodnost primárního slotu a spusť aplikaci.

V případě, že je MCUboot nakonfigurován pro podporou více aplikací, jsou výše zmíněné kroky provedeny pro každou aplikaci. Navíc je ještě kontrolována závislost mezi aplikacemi. [10]

3.3 Formát obrazu

Aby byl zavaděč schopný pracovat s firmwarem, je nutné aby jeho obraz měl předem definovaný formát. MCUboot definuje strukturu obrazu. Zjednodušeně řečeno, firmware je obalen hlavičkou a TLV záznamy. Struktura je znázorněna na obrázku 3.3. Společně se zavaděčem MCUboot je distribuován také nástroj `imgtool`, který z firmwaru vytvoří obraz ve zmíněném formátu. [10]

Obrázek 3.3 MCUboot - formát obrazu firmwaru [10]

3.3.1 Hlavička

Hlavička má velikost 32 bytů. Na jejím začátku se nachází magické slovo, podle kterého zavaděč pozná, že se v paměti nachází aplikace ve správném formátu. Dále je zde přítomná také velikost hlavičky (vyhrazené místo pro MCUboot hlavičku může být větší než 32 bytů), velikost firmwaru, verze firmwaru a další. Verze obrazu má tvar `<major>.<minor>.<revision>+<build_number>`. Kompletní struktura je vyobrazená na výpisu 3.1. [10]

Výpis kódu 3.1 MCUboot - struktura hlavičky [10]

```
#define IMAGE_MAGIC                 0x96f3b83d
#define IMAGE_HEADER_SIZE           32

struct image_version {
    uint8_t iv_major;
    uint8_t iv_minor;
    uint16_t iv_revision;
    uint32_t iv_build_num;
};

/** Image header.  All fields are in little endian byte order */
struct image_header {
    uint32_t ih_magic;
    uint32_t ih_load_addr;
    uint16_t ih_hdr_size;
    uint16_t ih_protect_tlv_size;
    uint32_t ih_img_size;
    uint32_t ih_flags;
    struct image_version ih_ver;
    uint32_t _pad1;
};
```

3.3.2 TLV záznamy

Na konci firmwaru se nacházejí TLV (type-length-value) záznamy. Tyto záznamy udávají další informace o firmwaru, například jak je firmware podepsán nebo šifrován. Také je zde uložen hash pro ověření integrity a pokud je používán secure boot nachází se zde také podpis. [10]

Rozlišují se dva typy TLV záznamů, obyčejné a chráněné. V chráněných TLV je typicky uložený podpis. U chráněných TLV je oproti obyčejným navíc ověřována integrita. Struktura TLV záznamu je vyobrazena na výpisu 3.2.[10]

Výpis kódu 3.2 MCUboot - struktura TLV záznamu [10]

```
#define IMAGE_TLV_INFO_MAGIC        0x6907
#define IMAGE_TLV_PROT_INFO_MAGIC   0x6908

/** Image TLV header.  All fields in little endian. */
struct image_tlv_info {
    uint16_t it_magic;
    uint16_t it_tlv_tot; /* size of TLV area */
};

/** Image trailer TLV format. All fields in little endian */
struct image_tlv {
    uint8_t  it_type;
    uint8_t  _pad;
    uint16_t it_len;
};
```

3.4 Nástroj imgtool

V repositáři projektu MCUboot se společně se zdrojovými kódy zavaděče nachází také nástroj `imgtool`. Nástroj slouží ke správě klíčů, podepisování firmware a vytváření obrazu ve formátu kompatibilním s MCUboot. [10]

Příkazem `imgtool.py keygen` je možné vygenerovat privátní klíč, pomocí kterého bude aplikace následně podepsána. MCUboot podporuje klíče RSA, ECDSA a EDDSA. Příkaz `imgtool.py getpub` extrahuje z privátního klíče veřejný klíč, výstupem je soubor v jazyce C, který obsahuje binární reprezentaci veřejného klíče. Veřejný klíč je následně zakompilován do firmwaru zavaděče. Všechny aplikace, se kterými bude v budoucnu zavaděč pracovat je následně nutné podepsat odpovídajícím privátním klíčem. Pomocí příkazu `imgtool.py sign` je možné vytvářet obraz firmwaru v předepsaném formátu. Výsledný soubor vždy obsahuje SHA256 hash, volitelně je také možné aplikaci podepsat. Privátní klíč, pomocí kterého se aplikace podepíše se specifikuje parametrem `--key`. Závislosti mezi aplikacemi mohou být specifikovány pomocí argumentu `-d "(image_id, image_version)"`, kde `image_id` je číslo aplikace, na kterém aktuální aplikace závisí, `image_version` je pak minimální potřebná verze specifikované aplikace. Například argument `-d "(1, 1.2.3+0)"` specifikuje závislost na aplikaci ve slotu číslo 1, minimální požadovaná verze je `1.2.3+0`. [10]

3.5 Port pro ESP32

MCUboot umí pracovat pouze s obrazem firmwaru, který je ve specifickém formátu popsaném v sekci 3.3. Formát firmwaru aplikace při použití zavaděče MCUboot se liší od standardního formátu pro platformu ESP32 popsaného v sekci 2.4. Struktura obrazu firmwaru je uvedena na obrázku 3.4.

Obrázek 3.4 MCUboot - struktura obrazu firmware aplikace pro ESP32 [10]

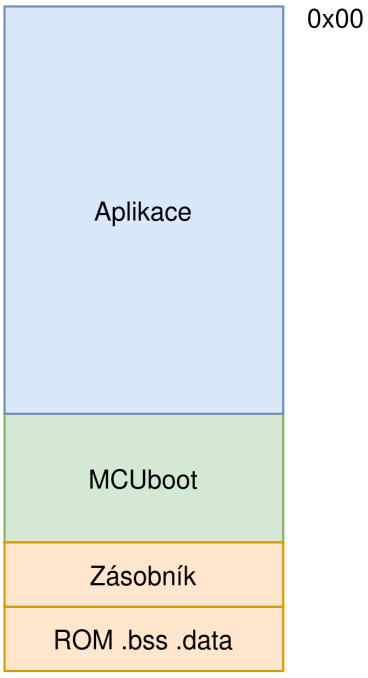
Na začátku firmware se nacházejí metadata, která obsahují data potřebná k nahrání aplikace do paměti a její spuštění. Struktura metadat je vyobrazena v tabulce 3.4. Po metadatech

následují data a instrukce aplikace, které se do paměti nahrávají případně mapují do adresního prostoru. Během linkování aplikace se počítá s tím, že prvních 32 bytů zabírá hlavička pro zavaděč MCUboot. Výsledný firmware má na začátku 32 bytovou výplň, která se během vytváření obrazu nástrojem `imgtool` nahradí hlavičkou. [10]

Tabulka 3.4 Struktura metadat [10]

offset	velikost (bytů)	popis
0	4	Magické slovo
4	4	Adresa vstupního bodu do aplikace
8	4	Adresa IRAM regionu v paměti RAM
12	4	Adresa IRAM regionu ve flash paměti relativní vůči slotu
16	4	Velikost IRAM regionu
20	4	Adresa DRAM regionu v paměti RAM
24	4	Adresa IRAM regionu ve flash paměti relativní vůči slotu
28	4	Velikost DRAM regionu

Po startu čipu je automaticky spuštěn zavaděč v paměti ROM. Tento zavaděč využívá část RAM paměti pro zásobník a pro data patřící do sekcí .bss (data inicializovaná na nulu) a .data (inicializovaná data). Zavaděč si během nahrávání budoucího firmware nesmí přepsat svou vlastní paměť. Paměť zavaděče MCUboot, který je využíván jako další zavaděč, proto nesmí kolidovat s pamětí ROM zavaděče. Stejně tak během zavádění aplikace nesmí MCUboot přepsat svou vlastní paměť. Rozdělení paměti je zobrazeno na obrázku 3.5. Některé čipy ESP32, například ESP32-S3 obsahují dvou-jádrový procesor. V takovém případě jsou v paměti vyhrazené oblasti pro zásobníky obou jader. [9] [10]



Obrázek 3.5 MCUboot - rozdělení paměti RAM [10]

Samotný zavaděč MCUboot je ještě rozdělen do tří paměťových regionů: `iram`, `iram_loader` a `dram`. Oproti standardní aplikaci jsou instrukce rozděleny do dvou regionů. Region `iram_loader`

obsahuje pouze zdrojový kód pro nahrání aplikace do paměti a její spuštění. Ostatní instrukce (logika aktualizací firmware, kryptografické knihovny, recovery, ...) jsou v regionu `iram`. Díky tomuto rozdělení je při nahrávání aplikace možné přepsat část zavaděče MCUboot, konkrétně region `iram`. Díky tomu má aplikace k dispozici více paměti pro statická data (ve výchozí konfiguraci jde o 32 KB). Po spuštění aplikace již paměť zavaděče MCUboot není využívaná a je možné jí celou využít pro haldu. [9] [10]

Kapitola 4

Operační systém Zephyr

Zephyr je open-source operační systém reálného času speciálně navržený pro využití ve vestavných systémech s omezenými zdroji a IoT zařízeních. Je vyvíjen jako open-source projekt pod záštitou Linux Foundation. Zephyr poskytuje robustní a flexibilní prostředí pro vývoj vestavěných systémů s různými požadavky na výkon, spotřebu energie a paměť. Operační systém podporuje širokou škálu architektur (ARM, RISC-V, Xtensa, MIPS, ...). Velkou předností systému je veliká modularita a hardwarová abstrakce. Pro začátečníky toto může zpočátku vypadat jako nevýhoda, ale pokud je aplikace správně napsaná, je velice jednoduché ji portovat na jinou architekturu. [13]

Na rozdíl od, dnes také velmi populárního systému, FreeRTOS není Zephyr jen plánovač, ale jde o celý ekosystém. Součástí je řada nástrojů, které usnadňují vývoj a testování. Aplikace je možné spouštět v QEMU, případně jako nativní aplikaci v operačním systému Linux. Dále systém obsahuje například terminál pro komunikaci se zařízením pomocí sériové linky a implementuje také několik způsobu pro aktualizace firmware. Systém nabízí bohatou podporu pro síťové připojení, obsahuje implementaci pro komunikaci pomocí TCP/IP a vlastní implementaci bluetooth kontroléru. Některé principy jsou převzaté z operačního systému Linux. Konkrétně definice hardwaru pomocí devicetree a konfigurace jádra pomocí Kconfig. Operační systém Zephyr kompatibilní s rozhraním POSIX. [13] [14] [15]

4.1 Jádro

Jádro je srdce každé aplikace využívající operační systém Zephyr. Poskytuje prostředí s podporou vícevláknového zpracování procesů a bohatou sadu dostupné funkcionality. Díky konfigurovatelnému jádru je možné do aplikace začlenit pouze potřebnou funkcionalitu a je tak ideální pro aplikace s omezenými zdroji. Při minimální konfiguraci jádra je požadavek na velikost paměti pouze 2 KB. [13]

4.1.1 Plánovač

Systém vybírá aktuálně spuštěné vlákno na základě jeho priority. Vlákno s nejvyšší prioritou má přednost. Pokud existuje více vláken se stejnou prioritou, systém vybere takové, které na zahájení čeká delší dobu. Priorita je celé číslo. Vlákno, které má nastavené nižší číslo priority má přednost před vlákny s vyšším číslem priority. [13]

4.1.2 Vlákna

Vlákno je nejmenší spustitelná jednotka operačního systému. Systém rozlišuje mezi dvěma typy vláken, kooperativním a preemptivním.

Kooperativní vlákno
Vlákno které je označeno jako kooperativní nemůže být plánovačem přerušeno, pokud k tomu samo nevykoná nějakou akci. Akcí vedoucí k přerušení může být například uspání vlákna, čekání na semafor nebo zavolání `k_yield()`.

Preemptivní vlákno
Vlákno označené jako preemptivní může být během svého běhu přerušené. Nejčastěji se tak děje pokud je připravené jiné vlákno s vyšší prioritou.

Každé vlákno může nabývat následujících třech stavů: běžící, připravené, neaktivní. Neaktivní je vlákno, které z nějakého důvodu nemůže běžet, například čeká na dokončení nějaké operace, bylo uspáno nebo ukončeno. Pokud chce být neaktivní vlákno spuštěno musí nejprve přejít do stavu připravené. V tomto stavu čeká, než ho plánovač vybere jako další aktivní vlákno. Když plánovač vlákno zvolí ke spuštění, je jeho stav nastaven na běžící. [13]

Systém dále dělí vlákna na uživatelská a systémová. Uživatelské je takové, které vytvořil uživatel, zatímco systémové je automaticky vytvořeno při startu systému. Systémová vlákna jsou minimálně dvě, vlákno `main` a `idle`. [13]

Vlákno `main` provádí nezbytnou inicializaci operační systému a spouští funkci `main()`. Po skončení funkce `main()` je vlákno ukončeno, systém ale stále běží. Vlákno `idle` je aktivní pouze pokud není připravené žádné jiné vlákno. Pokud to daná platforma podporuje, může být toto vlákno použito k přepnutí hardwaru do úsporného režimu. [13]

4.1.3 Pracovní fronta

Pracovní fronta je funkcionalita jádra, která využívá dedikované vlákno na zpracování pracovního objektu. Pracovní objekty jsou zpracovávány v pořadí v jaké byly do pracovní fronty zařazeny. Každý objekt pracovní fronty je zpracován tím, že je vykonána funkce, která příslušnému objektu náleží. Typicky je pracovní fronta využívána při obsluze přerušení nebo vláknem s vysokou prioritou pro odložené zpracování. Vlákno nebo obsluha přerušení vytvoří pracovní objekt a odešle ho do pracovní fronty. Pracovní objekt je následně zpracován na vlákně s nižší prioritou, až na něj přijde řada. [13]

4.1.4 Obsluha přerušení

Obsluha přerušení je funkce, která se spustí asynchronně na základě hardwarového nebo softwarového přerušení. Obsluha přerušení není plánována, aktuálně vykonávané vlákno je přerušeno. Díky tomu je zajištěna minimální režie během odezvy na přerušení. Vlákno je obnoveno po obsloužení všech přerušení. [13]

Jádro podporuje také vnořená přerušení. Tím je umožněno, aby byla aktuálně vykonávaná obsluha přerušena jiným přerušením s vyšší prioritou. Obsluha přerušení s nižší prioritou je obnovena poté, co je dokončena obsluha přerušení s vyšší prioritou. [13]

4.1.5 Uživatelský prostor

Jádro operačního systém Zephyr má také podporu pro uživatelský prostor. Tím je možné oddělit část paměti, kterou využívá jádro systému od paměti, kterou využívá aplikace. Hlavním cílem uživatelského režimu je poskytovat ochranu paměti a ochranu hardwaru před škodlivým nebo

chybným chováním softwaru. Pro podporu je požadováno aby cílová platforma měla implementované MPU. Platforma ESP32 obsahuje hardwarovou jednotku pro ochranu paměti, avšak tato funkcionalita zatím není v operačním systému Zephyr podporována. [5] [6] [13]

4.1.6 Ovladače

Standardní cestou, jak přistupovat k periferiím v operačním systému Zephyr je prostřednictvím ovladačů. Pro každý typ ovladače (UART, SPI, I2C, watchdog, časovač, ...) je v systému obecné rozhraní. S pomocí této abstrakce je možné psát aplikace nezávisle na platformě, na které budou spuštěny. Pro přístup k hardwaru aplikace se vyžívá jednotné rozhraní, které je identické napříč platformami. [13]

Každý ovladač je reprezentovaný strukturou `device`. Struktura obsahuje název ovládače (`name`), pointer na strukturu s konfigurací ovladače (`config`), pointer na data ovladače `data` a pointer na strukturu, která mapuje obecné rozhraní na konkrétní implementaci (`api`). [13]

Výpis kódu 4.1 Objekt reprezentující ovladač [13]

```
struct device {
      const char *name;
      const void *config;
      const void *api;
      void * const data;
};
```

Struktura `device` je vytvořena během inicializace ovladače. Pořadí v jakém jsou ovladače inicializovány je možné ovlivnit nastavením priority. [13]

Následující příklad odešle na sériovou linku znak `x`. Díky hardwarové abstrakci pomocí ovladačů bude příklad fungovat na všech platformách podporovaných v operačním systému Zephyr.

Výpis kódu 4.2 Příklad použití ovladače pro periferii UART

```
const struct device * dev = device_get_binding("uart0");
uart_poll_out(dev, 'x');
```

4.2 Kconfig

Kconfig je systém pro výběr konfigurace, který se běžně používá během sestavování projektu. Umožňuje povolit nebo zakázat funkcionalitu projektu. Původně byl vyvinut pro konfiguraci linuxového jádra, ale nyní se používá i v dalších projektech. [15]

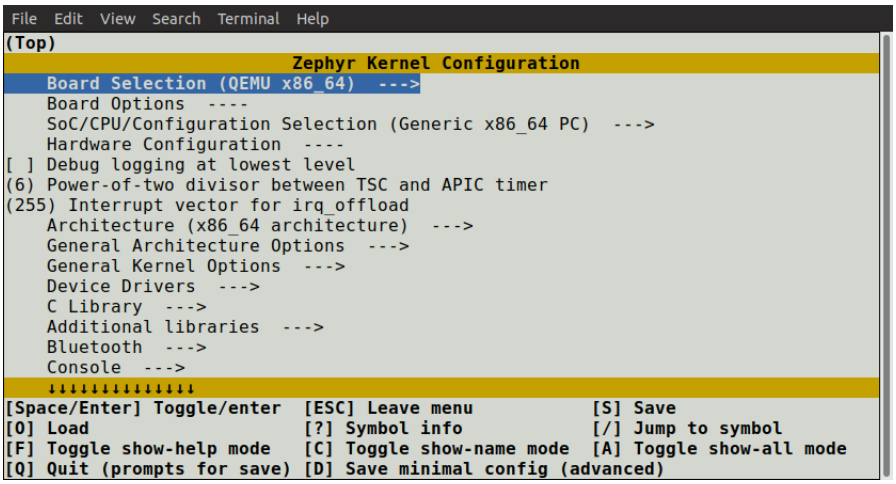
Operační systém Zephyr používá Kconfig pro konfiguraci funkcionality systému. Kconfig dovoluje konfigurovat projekt, aniž by bylo nutné upravovat zdrojový kód. Konfigurační volby (nazývané také symboly) jsou definovány v souborech `Kconifg`. V těchto souborech jsou také definované závislosti mezi konfiguračními volbami. Pro větší přehlednost jsou symboly seskupeny do víceúrovňové nabídky. [13] [15]

Konfiguraci je možné měnit interaktivně pomocí textového rozhraní `menuconfig` (obrázek 4.1), případně grafického `guiconfig` (obrázek 4.2) nebo je možné konfiguraci definovat přímo v souborech s příponou `.conf`. Výstupem konfigurace pomocí Kconfig je hlavičkový soubor `autoconf.h`, který obsahuje konfiguraci ve formě maker. Pomocí tohoto souboru je možné přistupovat ke konfiguraci ze zdrojového kódu. Na příkladu 4.3 je definice symbolu `FPU`, který přidává podporu pro operace v pohyblivé řádové čárce. Symbol je závislý na symbolu `CPU_HAS_FPU`. Symbol `FPU` je možné nastavit pouze v případě, že je aktivní také symbol `CPU_HAS_FPU`, v opačném případě mu zůstává výchozí hodnota. Symbol je možné nastavit pomocí `menuconfig`, `guiconfig` a nebo přidáním řádku `CONFIG_FPU=y` do souboru konfigurace projektu `prj.conf`.

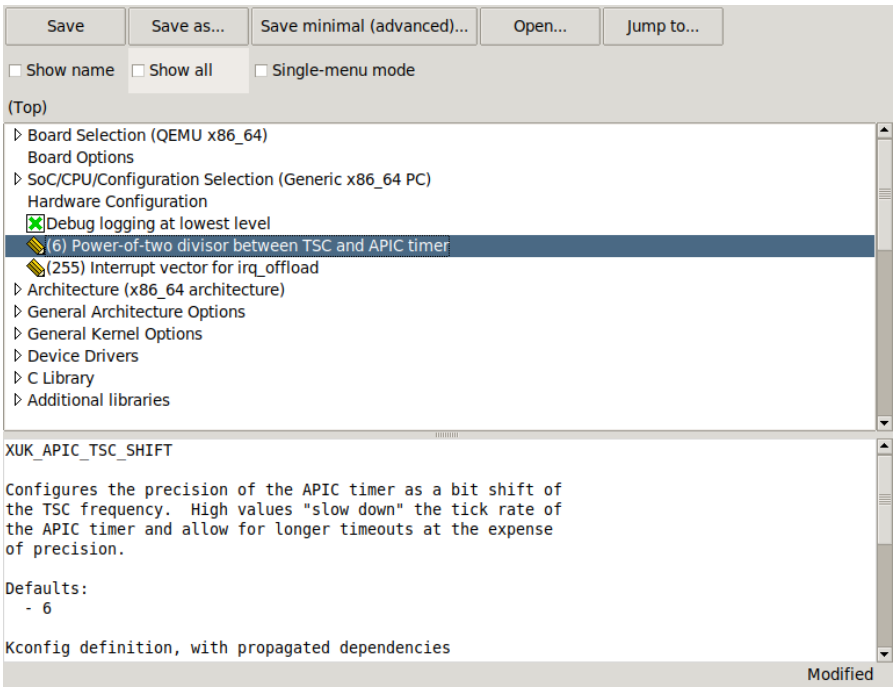
Výpis kódu 4.3 Kconfig - příklad definice konfiguračních symbolů [13]

```
config CPU_HAS_FPU
   bool
   help
     This symbol is y if the CPU has a hardware floating point unit.

config FPU
   bool "Support floating point operations"
   depends on HAS_FPU
```



Obrázek 4.1 Konfigurace Kconfig pomocí menuconfig [13]



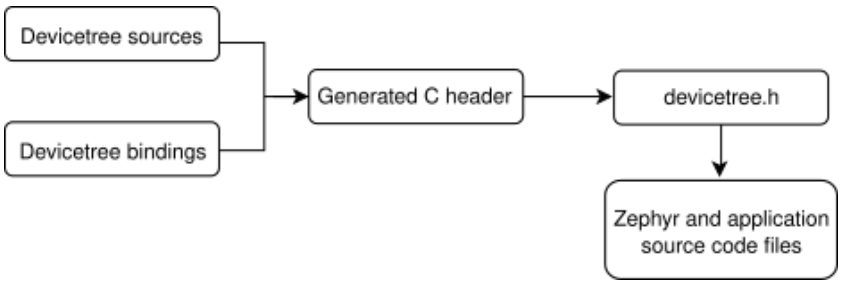
Obrázek 4.2 Konfigurace Kconfig pomocí guiconfig [13]

4.3 Devicetree

V operačním systému Zephyr se devicetree používá k popisu hardwarové konfigurace systému. Jedná se o stromovou datovou strukturu, která definuje zařízení, jejich vlastnosti a jejich vzájemné propojení. Devicetree je deklarativní formát, což znamená, že definuje požadovanou strukturu konfigurace, nikoli to, jak ji dosáhnout. Devicetree umožňuje jednoduchou konfiguraci hardwaru a její přizpůsobení různým platformám, což je zásadní pro přenositelnost a flexibilitu operačního systému. [13] [16]

Operační systém Zephyr používá devicetree pro popis hardwaru na podporovaných deskách a pro popis počáteční hardwarové konfigurace, Zephyr jej tedy používá jako jazyk pro popis hardwaru a částečně také konfiguraci systému (pomocí devicetree je například nastavena výchozí sériová linka, která se používá pro systémový terminál). [13]

Způsob, jakým operační systém využívá devicetree se značně liší od linuxu. V operačním systému linux jsou zdrojové soubory devicetree převedeny do binární podoby, která je načtena během startu systému. To dovoluje dynamicky měnit konfiguraci hardwaru za běhu systému. Operační systém Zephyr oproti tomu převádí zdrojový kód devicetree do hlavičkového souboru, ve kterém se informace z devicetree nacházejí v podobě maker. Aplikace využívající Zephyr je vždy sestavena pro konkrétní hardwarovou konfiguraci, není ji možné dynamicky měnit za běhu systému. Způsob překladu devicetree je vyobrazen na obrázku 4.3. [13] [14]



Obrázek 4.3 Zpracování devicetree v RTOS Zephyr [13]

Samotná konfigurace pomocí devicetree se v RTOS Zephyr skládá ze dvou částí, devicetree bindings a zdrojového souboru. Devicetree bindings popisují strukturu uzlů, které se používají ve zdrojovém kódu. Jedná se o soubor ve formátu YAML, který definuje jaké má uzel parametry a zda může být vnořený do jiného uzlu. Příklad 4.4 popisuje strukturu uzlu definujícího sériové rozhraní. Na dalším příkladu 4.5 je znázorněno použití uzlu pro sériové rozhraní. [13] [16]

Výpis kódu 4.4 Příklad devicetree bindings [13]

```
compatible: "manufacturer,serial"

properties:
  reg:
    type: array
    description: UART peripheral MMIO register space
    required: true
  current-speed:
    type: int
    description: current baud rate
    required: true
```

Výpis kódu 4.5 Příklad zdrojového souboru devicetree [13]

```
/ {
    soc {
        uart0@40003000 {
            compatible = "manufacturer,serial";
            reg = <0x40003000 0x1000>;
            current-speed=<115200>
        };
    };
};
```

4.4 Nástroj west

Operační systém Zephyr je závislý na externích knihovnách. Mezi externí projekty patří například zavaděč MCUboot, kryptografická knihovna Mbed TLS nebo vrstva hardwarové abstrakce pro platformu ESP32, `hal_espressif`, kterou spravuje samotná firma Espressif System a jsou pomocí ní implementované ovladače. Nástroj `west` umožňuje správu pracovního adresáře, správu externích projektů, jejich aktualizaci a řeší závislostí mezi nimi. Kromě toho poskytuje příkazy pro sestavení projektu, nahrávání firmware do zařízení a ladění. [13]

4.5 Build systém

Operační systém používá pro sestavení aplikace a jádra nástroj CMake. Sestavení pomocí CMake sestává ze dvou fází, konfigurační fáze a fáze sestavení. V konfigurační fázi jsou zpracovány všechny soubory `CMakeLists.txt`. Během toho jsou nastaveny cesty ke zdrojovým souborům a hlavičkovým souborům, kromě toho jsou také zpracovány konfigurační soubory devicetree a Kconfig. Výstupem konfigurační fáze jsou skripty pro nástroje `Make` nebo `Ninja`, které provedou sestavení projektu. Výsledkem je zkompilovaný projekt ve formátu ELF, se kterým je možné dále pracovat. Ze souboru ve formátu ELF jsou pomocí nástroje `objdump` vytvořeny soubory ve formátu `.bin` a nebo `.hex`. Tyto soubory je možné ještě dále zpracovávát, pokud jde o aplikaci určenou pro zavaděč MCUboot může být binární soubor následně ještě podepsán. Na platformě ESP32 se firmware, který je zaváděný zavaděčem v paměti ROM, vytváří ze souboru ELF pomocí nástroje `elf2image`. [9] [13]

Od verze 3.4 se v RTOS Zephyr nachází rozšíření sysbuild, které dovoluje kombinovat několik build systémů do jednoho. Například lze sestavit aplikaci společně se zavaděčem a oba tyto projekty najednou nahrát do zařízení. Sysbuild zároveň poskytuje CMake API pro rozšíření funkcionality. Je možné například sjednotit binární soubory ze všech projektů do jednoho binárního souboru. [13]

4.6 Aktualizace firmware

Jak již bylo zmíněno na začátku kapitoly, operační systém Zephyr nabízí v základu mnoho funkcionality. Jedním ze stěžejních funkcí, která vývojářům usnadňuje mnoho práce, je podpora pro aktualizace firmware. Způsob aktualizací je dokonce stejný na jakémkoliv podporovaném hardwaru. V praxi to znamená, že například SoC platformy ESP32 se aktualizuje stejným způsobem jako SoC z rodiny STM32 nebo nRF52. [13]

4.6.1 Podpora pro MCUboot

Zavaděč MCUboot je jedním ze základních kamenů pro podporu aktualizací firmware, zároveň je výchozím zavaděčem pro operační systém Zephyr. Port zavaděče MCUboot je napsaný jako aplikace v operačním systému Zephyr. Díky tomu může využívat již existující ovladače periferií. Zároveň používá další funkcionalitu, kterou Zephyr nabízí, jako je Kconfig, devicetree a stejný build systém. Každá deska specifikuje rozložení paměti pomocí devicetree. Výpis kódu 4.6 zobrazuje rozložení flash paměti na desce `esp32c3_devkitm`. MCUboot využívá právě definice z devicetree k rozdělení flash paměti na sloty. [10] [13]

Výpis kódu 4.6 Definice rozložení paměti při použití zavaděče MCUboot [13]

```
&flash0 {
        status = "okay";
        partitions {
                compatible = "fixed-partitions";
                #address-cells = <1>;
                #size-cells = <1>;

                /* Reserve 60kB for the bootloader */
                boot_partition: partition@0 {
                        label = "mcuboot";
                        reg = <0x00000000 0x0000F000>;
                        read-only;
                };

                /* Reserve 1024kB for the application in slot 0 */
                slot0_partition: partition@10000 {
                        label = "image-0";
                        reg = <0x00010000 0x00100000>;
                };

                /* Reserve 1024kB for the application in slot 1 */
                slot1_partition: partition@110000 {
                        label = "image-1";
                        reg = <0x00110000 0x00100000>;
                };

                /* Reserve 256kB for the scratch partition */
                scratch_partition: partition@210000 {
                           label = "image-scratch";
                           reg = <0x00210000 0x00040000>;
                };

                storage_partition: partition@250000 {
                        label = "storage";
                        reg = <0x00250000 0x00006000>;
                };
        };
};
```

Konfigurační symbol `CONFIG_BOOTLOADER_MCUBOOT` indikuje, že je aplikace určena k zavedení zavaděčem MCUboot. V rámci sestavování aplikace dojde také k vytvoření obrazu ve správném formátu pomocí nástroje `imgtool`. Výstupem je binární soubor který je možné nahrát přímo do zařízení a nebo ho použít jako soubor pro aktualizaci firmware. Samotný MCUboot je možné sestavit dvěma způsoby, odděleně jako jakoukoliv jinou aplikaci nebo společně s aplikací pomocí rozšíření sysbuild. [10] [13]

4.6.2 Nástroj MCUmgr

MCUmgr je nástroj pro vzdálenou správu vestavných zařízení. Nástroj je modulárně koncipovaný, jednotlivé moduly poskytují různou funkcionalitu. V základu nástroj obsahuje moduly pro správu souborového systému, přístup k terminálu zařízení, aktualizace firmwaru a další. Aktualizace firmwaru jsou kompatibilní se zavaděčem MCUboot. Se zařízeními je možné komunikovat pomocí sériového rozhraní, bluetooth nebo po sítí pomocí UDP. [17]

MCUmgr sestává ze dvou částí, klienta a SMP serveru. Klientem je typicky počítač nebo telefon, pomocí kterého se zařízení spravuje. Na straně zařízení pak musí být implementován SMP server. MCUmgr lze také používat pro komunikaci mezi dvěma mikroprocesory, v takovém případě jedno zařízení funguje jako klient a na druhém je aktivní server. Operační systém Zephyr obsahuje implementaci klienta i SMP serveru. [13] [17]

Pro PC existuje klient MCU Manager CLI, prostřednictvím kterého lze komunikovat se zařízeními obsahující SMP server. Na příkladu 4.7 je ukázáno vytvoření připojení prostřednictvím sériového rozhraní a nahrání aktualizace do zařízení. Podpora pro MCUmgr v operačním systému Zephyr využívá definice rozdělení flash paměti z devicetree pro zjištění do jakého slotu má být aktualizace nahrána.

Od firmy Nordic Semiconductor je k dispozici mobilní aplikace nRF Connect Device Manager, která implementuje klienta pro mobilní zařízení. Komunikace se zařízením je realizována pomocí technologie bluetooth. [17] [18]

Výpis kódu 4.7 Aktualizace firmware pomocí MCUmgr

```
# Create connection
$ mcumgr conn add acm0 type="serial"  \
    connstring="dev=/dev/ttyACM0,baud=115200,mtu=512"

# upload image to secondary slot
$ mcumgr -c acm0 image upload <binary file>

# confirm update
$ mcumgr -c acm0 image confirm <hash>

# restart device
$ mcumgr -c acm0 reset
```

Kapitola 5
Návrh řešení

Ve firmě, kde autor pracuje, se dosud používaly procesory z řady nRF52 a nRF91 od výrobce Nordic Semiconductor. Nově se připravují produkty, které vyžadují Wi-Fi konektivitu. Právě kvůli zmíněné Wi-Fi konektivitě budou nová zařízení obsahovat procesory z rodiny ESP32. Způsob jakým jsou vyřešené aktualizace firmwaru umožňuje kromě aktualizace aplikace, také aktualizace zavaděče. Je žádoucí aby aktualizace zařízení fungovali stejně napříč různými produkty. Důvodů proč to chtít je hned několik. Jedním z nich je, že je MCUboot upravený, recovery podporuje firemní proprietární sériový protokol pro komunikaci mezi několika mikroprocesory. Pro delší podporu produktu je vhodné mít možnost protokol aktualizovat. Dalším důvodem může být delší podpora pro existující produkty, v budoucnu se může stát, že dojde ke změnám v projektu MCUboot a operačním systému. Díky tomu, že lze zavaděč aktualizovat je možné aby všechny produkty používaly aktuální verzi operačního systému Zephyr.

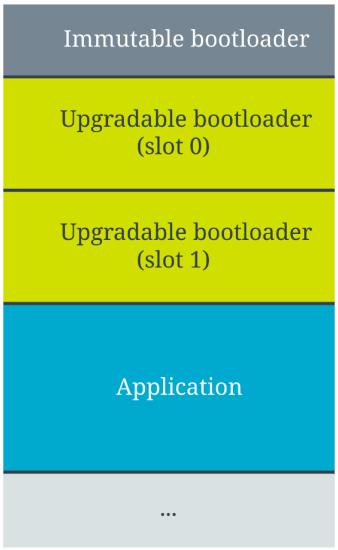
5.1 Konkurenční řešení

Firma Nordic Semiconductor své NRF Connect SDK staví nad operačním systémem Zephyr. Jako zavaděč je využívána jejich proprietární implementace nebo je možné využít zavaděč MCUboot. V kombinaci s proprietárním zavaděčem je možné MCUboot použít jako druhý aktualizovatelný zavaděč. [11]

Pří použití MCUboot jako druhého aktualizovatelného zavaděč má MCUboot ve flash paměti vyhrazené dva slot. Z jednoho je spouštěn a do druhého patří nová aktualizovaná verze. První zavaděč se na základě obsahu těchto dvou slotů rozhodne, který MCUboot spustí. Rozhoduje se na základě verze, spuštěn je ten, který má vyšší verzi. [11]

Proces aktualizace může vypadat následovně. Ve výchozím stavu se ve slotu 0 nachází MCUboot s verzí 1, slot 1 je prázdný. První zavaděč spustí MCUboot ve slotu 0. Pro aktualizaci se nahraje nový MCUboot verze 2 do slotu 1 a zařízení se restartuje. V tuto chvíli první zavaděč vidí dva firmwary, ve slotu 0 je verze 1 a ve slotu 1 verze 2. Zavaděč spustí firmware s vyšší verzí, tedy MCUboot ve slotu 1. V tuto chvílí je aktivní MCUboot ve slotu 1, další aktualizace musí být nahrána do slotu 0. Stejným způsobem se při dalších aktualizacích mění slot, ze kterého je MCUboot spuštěn. [10] [11]

Aktualizace zavaděče i aplikace řeší sám MCUboot. Je využita jeho podpora pro více aktualizovatelných aplikací. Aplikace je aplikace 0 a zavaděč MCUboot aplikace 1. Každá aplikace má svůj primární slot (u MCUboot je to ten, ze kterého není spuštěn). Sekundární slot je jen jeden, sdílený. Toto standardně MCUboot neumožňuje. Společně s NRF Connect SDK je také distribuována verze projektu MCUboot s potřebnými úpravami. [11]



Obrázek 5.1 NRF Connect SDK - rozdělení flash paměti při aktualizovatelném zavaděči [11]

Popsaný způsob jakým funguje proprietární zavaděč z NRF Connect SDK není jedinečný pouze pro toto řešení. Stejným způsobem funguje zavaděč MCUboot, pokud je nakonfigurován v režimu Direct-XIP. Podobně jsou řešené aktualizace také na platformě ESP32 při použití zavaděče z ESP-IDF SDK. Opět se mezi dvěma sloty vybírá jaká verze bude spuštěna. Řešení z ESP-IDF SDK však nelze použít, protože je nekompatibilní se způsobem aktualizací v operačním systému Zephyr. [9] [10]

5.2 První zavaděč

Pro spolehlivé a odolné aktualizace zavaděče je nutné mít v systému minimálně dva zavaděče. První minimální, který již nebude možné aktualizovat a druhý, aktualizovatelný. Aby byla aktualizace zavaděče bezpečná nikdy nesmí dojít k přepsání oblasti flash paměti, ve které se zavaděč nachází. Pokud by během zápisu do oblasti zavaděč došlo k výpadku napájení, v zařízení nebude existovat validní firmware zavaděče a zařízení již nebude možné spustit. Pro zavaděč MCUboot budou v paměti alokovány dva sloty. MCUboot může být spuštěn z obou slotů. Na základě verze se rozhodne jaký zavaděč bude spuštěn. Aktualizace bude nahrávána do druhého slotu (takového ze kterého zrovna není zavaděč spuštěn). Pokud by aktualizace neproběhla úspěšně v jednom slotu bude nevalidní firmware a v druhém bude stále starý zavaděč MCUboot, který bude spuštěn.

Tím se většina práce redukuje na implementaci prvního zavaděče a použití MCUboot jako druhého zavaděče. V této práci může být první zavaděč označen také anglickým výrazem `immutable bootloader`. Princip, jakým bude fungovat první zavaděč, bude stejný jako je popsáno v sekci 5.1. První zavaděč bude mezi dvěma sloty vybírat, jakou verzi druhého zavaděče MCUboot spustí. Společně s tím je třeba vyřešit následující věci:

- Napsání prvního zavaděče.
- Použití MCUboot jako druhého zavaděče.
- Přesunutí počáteční inicializace hardwaru do prvního zavaděče.
- Do firmwaru e třeba zakomponovat jeho verzi.

- Rozdělení paměti flash a RAM.
- Sestavení dvou variant zavaděče MCUboot pro dva odlišné sloty.
- Podpora pro secure boot a s tím související způsob uchovávání klíčů.
- Podepisování firmware.

5.3 Aktualizace zavaděče

Aktualizace zavaděče bude probíhat podobně jako aktualizace aplikace. Zavaděč MCUboot dokáže aktualizovat více firmware. Jedním z nich bude aplikace a druhým sám zavaděč MCUboot. Každý aktualizovaný firmware má definované dva sloty, primární a sekundární. Nový firmware se nahrává do sekundárního. Až je MCUboot spuštěn, detekuje v sekundárním slotu nový firmware a provede aktualizaci do primárního slotu. Původně bylo v plánu používat MCUboot standardní cestou, to by znamenalo, že aplikace i zavaděč budou mít svůj vlastní sekundární slot. Díky tomuto řešení by nebylo nutné provádět žádné změny v zavaděči MCUboot a také by řešení umožňovalo aktualizovat aplikaci a zavaděč současně. Bohužel způsob jakým jsou v operačním systému Zephyr konfigurovány sloty jednotlivých aplikací nedovoluje dynamicky měnit primární slot pro zavaděč MCUboot a toto řešení nemůže být použito.

Bude tedy nutné aplikovat do projektu MCUboot změny, které se nacházejí v NRF Connect SDK. Soubory, ve kterých se nacházejí změny jsou distribuovány pod licencí Apache 2.0, takže není problém je využít také pro čipy ESP32. NRF Connect SDK využívá jeden sdílený slot pro aplikaci i zavaděč. V operačním systému Zephyr je definován pouze primární a sekundární slot pro aplikaci. To ničemu nevadí, protože sekundární slot je sdílený také pro zavaděč. Po nahrání firmware do sekundárního slotu následně MCUboot detekuje, zda jde o aplikační firmware, či zavaděč a provede aktualizaci do správného primárního slotu. Více informací o rozdělení slotů je sekci 5.7.

5.4 Formát obrazů firmware

Na platformě ESP32 je po startu spuštěn zavaděč v paměti ROM tento zavaděč očekává firmware ve specifickém formátu (obrázek 2.3) na adrese `0x00` ve flash paměti. Z toho vyplývá, že první zavaděč musí dodržet požadovaný formát. Obraz tedy bude vytvářen nástrojem `elf2image`.

MCUboot již následně není zaváděn ROM zavaděče, nemusí dodržovat předepsaný formát. Vzhledem k tomu, že aktualizaci bude provádět sám MCUboot je nutné aby byl výsledný obraz firmware vytvořený nástrojem `imgtool`. Z tohoto důvodu bude formát vypadat stejně jako firmware aplikace, tj. stejně jako na obrázku 3.3.

5.5 Struktura popisující firmware

První zavaděč musí být schopný zjistit jaká verze firmware se ve slotu nachází. K tomuto účelu bude firmware zavaděče MCUboot obsahovat strukturu `img_info`. Struktura se bude bude nacházet na začátku firmware, hned za metadaty (obrázek 5.2). Díky tomu bude vždy na fixní adrese a předchozí zavaděč ji může přečíst. Firmware zavaděče MCUboot bude na začátku firmware obsahovat volné místo pro MCUboot hlavičku, typicky 32 bytů. Za MCUboot hlavičkou se budou nacházet metadata, která mají velikost také 32 bytů. Následovat bude struktura `img_info`, která bude začínat na 64. bytu. Pokud se bude firmware nacházet například na adrese `0x20000`, struktura s popisem firmwaru se nachází na adrese `0x20040`.

Struktura obsahuje magické slovo, podle kterého je možné rozpoznat, že se v paměti opravdu nachází tato struktura. Následuje verze struktury (v budoucnu je možné ji rozšiřovat) a velikost

Obrázek 5.2 Umístění struktury `img_info`

struktury. Dále jsou zde obsaženy informace o firmwaru, jako je jeho verze a velikost (včetně metadata a této struktury). Adresa flash paměti, pro kterou je firmware určen. Pomocí ní je možné detekovat, zda se firmware nenachází ve špatném slotu, což by mělo za následek pád při spuštění firmwaru a mohlo by vést k znefunkčnění zařízení. Poslední položkou je značka, zda je firmware validní.

V této struktuře není uložena žádná informace, pomocí které by bylo možné kontrolovat integritu firmwaru. Toto rozhodnutí je učiněno úmyslně. Na produkci bude vždy aktivovaný secure boot a firmware bude tedy podepsaný. Při ověřovaní podpisu se kontroluje také integrita firmwaru. Zároveň je integrita firmware kontrolována zavaděčem MCUboot předtím, než je provedena aktualizace. Nástroj `imgtool` automaticky přidává do obrazu firmware jeho hash.

Tabulka 5.1 Obsah struktury `img_info`

offset	velikost (bytů)	popis
0	4	Magické slovo
4	2	Verze struktury
6	2	Velikost struktury
8	4	Verze firmware
12	4	Velikost firmware (včetně metadat a této struktury)
16	4	Adresa firmware ve flash paměti
20	4	Značka, zda je firmware validní

Slot, ze kterého je zavaděč MCUboot aktuálně spuštěn je možné rozeznat podle verze a informace o tom, zda je validní. Nevalidní firmware je například firmware, který má neplatný podpis, nebo je určen do jiného slotu, než ve kterém se nachází. V případě, že firmware není validní, první zavaděč jej zneplatní zapsáním specifické hodnoty do flash paměti na místo validní značky. Značka, zda je firmware validní, dovoluje v kombinaci s verzí aplikaci zjistit, v jakém slotu je MCUboot právě aktivní, toho může být využito při aktualizaci zavaděče. Před aktualizací bude možné zjistit jakou variantu MCUboot je potřeba do zařízení nahrát, zda variantu pro slot 0 nebo 1.

5.6 Rozdělení paměti RAM

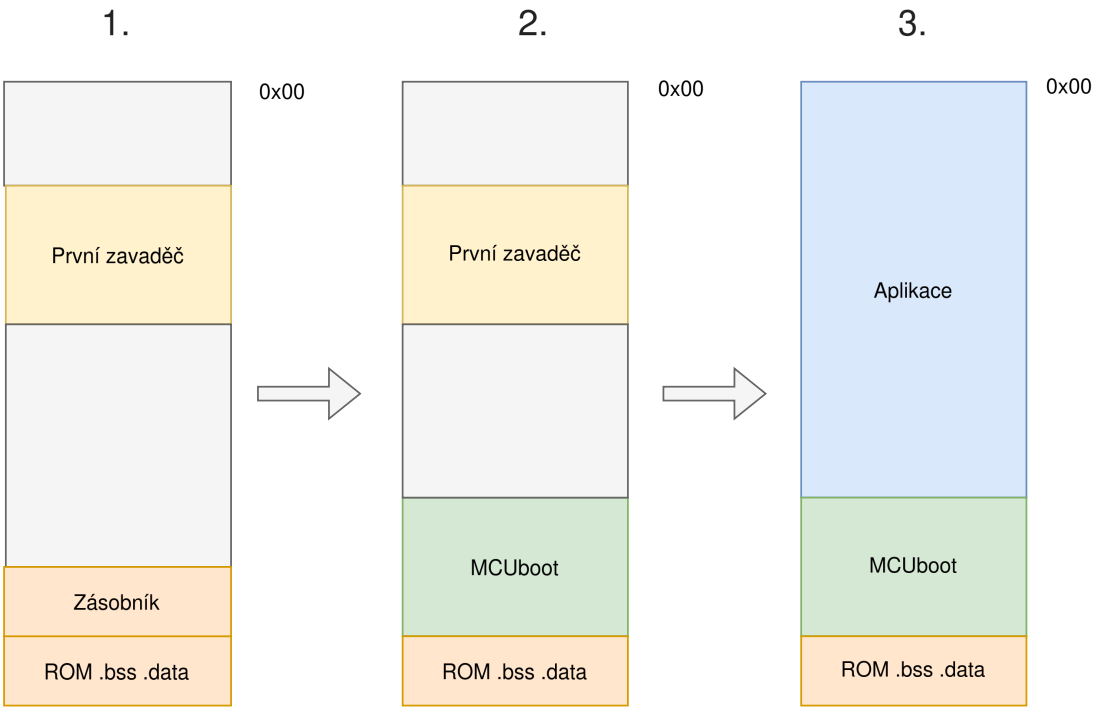
Po spuštění čipu platformy ESP32 je spuštěn zavaděč v paměti ROM. část paměti RAM je využívána tímto zavaděčem. Na konci paměti je vyhrazené místo pro zásobník a data. Pří zavádění firmware je třeba dávat pozor aby nedošlo k přepsání některých z těchto paměťových sekcí. Část dat je navíc využívána kódem uloženým v ROM paměti i během běhu aplikace. Tuto paměť není možné nikdy přepsat.

Cílem je, aby měla aplikace co nejvíce místa dostupného pro statická data, z toho vyplývá, že zavaděč může být umístěný na začátku nebo na konci paměti. MCUboot nesmí během zavádění aplikace přepsat část své paměti. Nejvýhodněji se jeví jeho umístění na konec paměti.

První zavaděč je umístěn v první polovině paměti ROM. Zavaděč je možné umístit také na samotný začátek, takové řešení bude ovšem fungovat jen do té doby, než se aktivuje secure boot. S aktivovaným secure boot není ROM zavaděč schopný zavést firmware na začátek paměti. Tento požadavek byl zjištěn čistě náhodou. Dokumentace nikde nezmiňuje nic o využiti počátku

paměti. Bohužel zůstává neznámo jak velkou část paměti není možné obsadit. Bylo vyzkoušeno, že pokud prvních 128 KB zůstane volných je firmware v pořádku spuštěn.

První zavaděč má za úkol zavést MCUboot, obě varianty MCUboot jsou linkovány na stejné adresy. V době kdy běží první zavaděč bude zásobník nastaven na místo nacházející se uvnitř paměti vyhrazené pro první zavaděč. MCUboot může v tuto chvíli obsadit i část paměti, kterou ROM zavaděč používal jako zásobník. Po spuštění druhého zavaděče MCUboot již část paměti, kterou využíval první zavaděč, není využívána a je možné ji přepsat aplikací. Využití paměti během závádění je znázorněno na obrázku 5.3.

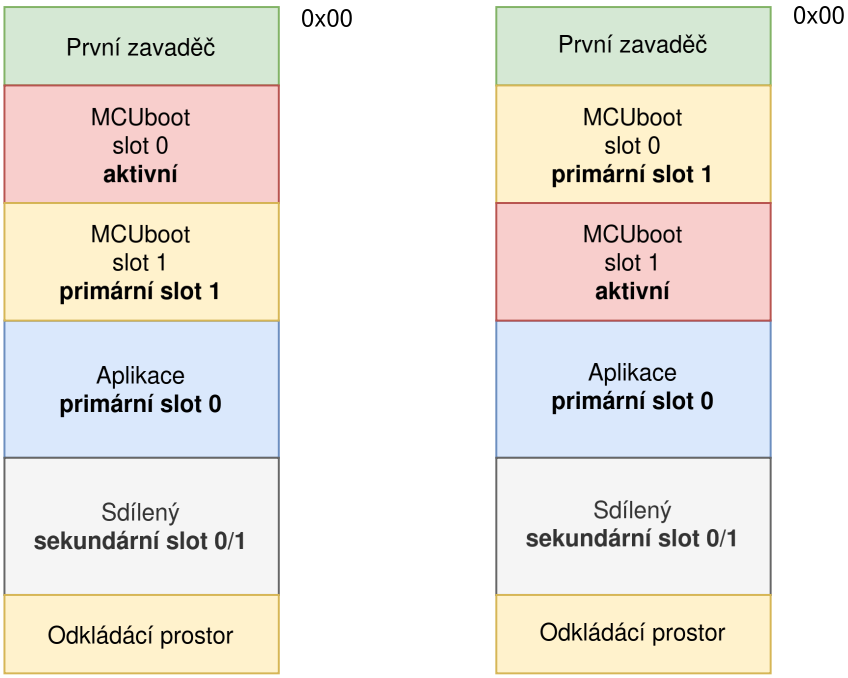


Obrázek 5.3 Rozdělení paměti RAM během zavádění aplikace

Výše zmíněné rozložení je platné pro čip ESP32-C3, Pro ostatní čipy se může lišit. Některé čipy obsahují dvou-jádrový procesor a část paměti je navíc vyhrazená pro zásobník druhého jádra. Rozdělení paměti je vždy nutné přizpůsobit požadavkům konkrétního čipu. Záměrně zde nejsou uvedeny adresy regionů, protože závisí na velikosti jednotlivcích paměťových regionů. Ty se mohou v závislosti na konfiguraci zavaděče MCUboot měnit.

5.7 Struktura flash paměti

Zavaděč v paměti ROM očekává firmware ve flash paměti na adrese `0x00`. První zavaděč musí být umístěn na začátku flash paměti. Následují dva sloty pro zavaděč MCUboot, mezi kterými si první zavaděč vybírá. Po zavaděčích se nachází primární slot pro aplikaci a následuje sdílený sekundární slot. Zavaděč MCUboot bude nakonfigurován pro aktualizace pomocí odkládacího prostoru, v paměti je nutné mít vyhrazené místo pro odkládací prostor. Takto uvedená struktura je pouze doporučení, není nutné ji dodržovat. Adresy slotů je možné nastavit libovolně, ani není nutné aby byly v tomto pořadí. Jen nutné aby první zavaděč byl na adrese `0x00` a firmware, který je sestavený v režimu XIP (mapuje flash paměť do adresního prostoru) musí být zarovnaný na velikost stránky, což je 64 KB.



Obrázek 5.4 Rozdělení paměti flash z pohledu MCUboot ve slotu 0 a 1

Pro oba sloty 0 a 1 bude nutné sestavit odlišné firmwary zavaděče MCUboot, taky aby měli nastavené správně primární a sekundární sloty. Varianty se liší v nastavení primárního slotu pro zavaděč MCUboot. Varianta, která je určena pro slot 0 má primární slot pro zavaděč nastavený na slot 1, druhá varianta naopak. Na obrázku 5.4 je vyobrazeno rozdělení flash paměti včetně primárních a sekundárních slotů. Vlevo je varianta, kdy je MCUboot spuštěn ze slotu 0 a vpravo ze slotu 1. Primární slot pro aplikaci se nijak neliší. Aplikaci náleží primární a sekundární slot 0, zavaděči MCUboot primární a sekundární slot 1.

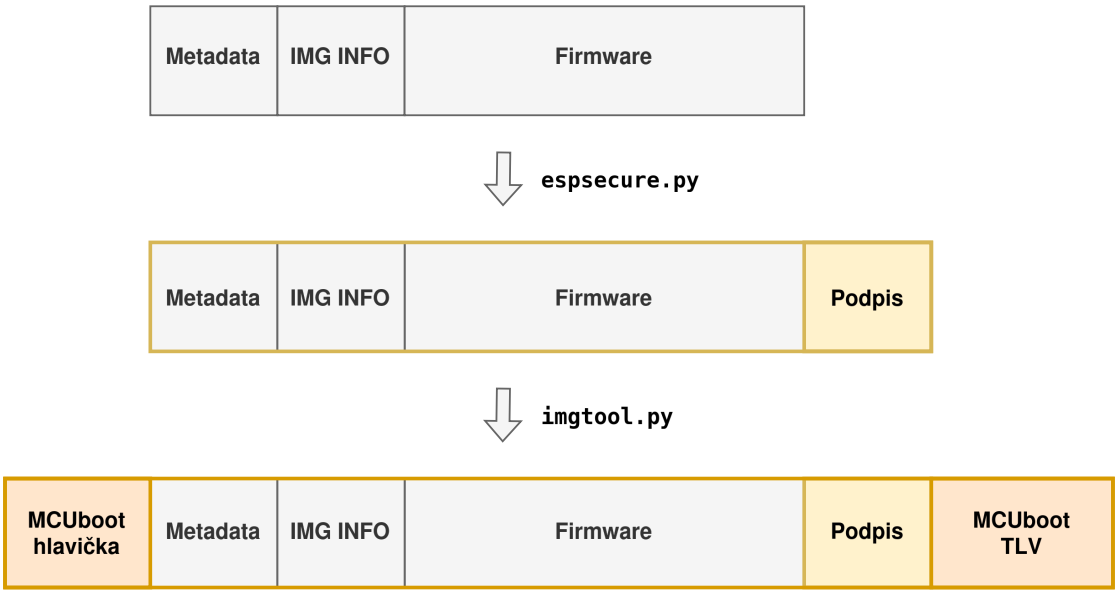
5.8 Secure boot

Aby byl secure boot účinný je nutné ověřovat každý zaváděný firmware. To znamená ověřovat první zavaděč, MCUboot a aplikaci. Zavaděč v paměti ROM není třeba ověřovat, není možné jej nijak přepsat. Zavaděč v ROM paměti podporuje secure boot (více informací je v kapitole 2). Zavaděč MCUboot má také implementovaný secure boot, pomocí kterého ověřuje aplikaci. Zbývá implementovat secure boot do prvního zavaděče, který bude ověřovat firmware druhého zavaděče MCUboot.

Existuje několik způsobů, jak implementovat secure boot. Bylo by možné využít některou z kryptografických knihoven k ověřování podpisu a napsat vlastní skript, který firmware podepíše. Toto řešení je náročnější a také pomalejší. Na všech čipech z rodiny ESP32 je dostupný secure boot a v ROM paměti se nacházejí kryptografické funkce, které lze volat. Takovou funkcí je například `ets_rsa_pss_verify`, která ověří podpis firmwaru. Stačí ji na vstupu dodat veřejný klíč, který se nachází v podpisu, celý podpis, a spočítaný SHA256 hash firmware. Tím, že se funkce nachází v paměti ROM nezabírají v prvním zavaděči žádnou paměť navíc. Funkce využívají hardwarovou akceleraci. Formát podpisu musí být ve formátu, v jakém ho vytváří nástroj `espsecure.py`. K podepisování bude tedy využit tento nástroj. Binární soubor na vstupu nebude obsahovat výplň pro MCUboot hlavičku. Po nahrazení výplně hlavičkou by byl podpis nevalidní.

Podpis je ověřen pomocí přiloženého veřejného klíče, který se nachází v podpisovém bloku. Tento klíč bude nutné nejprve ověřit, zda je důvěryhodný. Hash veřejného klíče bude uložen v eFuse a před ověřením podpisu vždy dojde také k ověření důvěryhodnosti veřejného klíče.

Firmware zavaděče MCUboot bude ještě nutné podepisovat pomocí `imgtool.py`. Samotný zavaděč MCUboot před aktualizací také ověřuje podpis. První zavaděč se na firmware dívá počínaje metadaty, vůbec ho tedy nezajímá MCUboot hlavička (stejně tak TLV). Podpis pomocí `imgtool.py` slouží čistě pro účely aktualizace. Prvnímu zavaděči jen stačí, že se firmware nachází na správném místě a je podepsán nástrojem `espsecure.py`. Způsob, jakým se bude firmware zavaděče MCUboot podepisovat je vyobrazen na obrázku 5.5. Aplikace bude podepsána standardně pouze pomocí `imgtool.py` a první zavaděč pouze pomocí `espsecure.py`.



Obrázek 5.5 Způsob podepisování firmwaru zavaděče MCUboot

5.9 Limitace řešení

Použití sdíleného sekundárního slotu pro umístění aktualizací standardně MCUboot nepodporuje. V závislosti na aktuálně aktivním zavaděči MCUboot se také mění jeho primární slot. To sebou nese jistá omezení, která je třeba mít na paměti.

- Nelze provést testování aktualizace pří aktualizaci zavaděče MCUboot
- Režim aktualizací Direct-XIP a RAM load není podporován
- Aplikace nezná informace o primárním slotu zavaděče MCUboot

Během testování aktualizace zavaděč MCUboot provede nahrání firmware do primárního slotu. Pokud nedojde k potvrzení firmware je následně vrácena původní verze, která se nachází v sekundárním slotu. Pokud dojde k aktualizaci zavaděče MCUboot, jeho nová verze je spuštěna z odlišného slotu a zároveň je změněný primární slot, do kterého patří zavaděč. V případě vrácení původní verze by došlo k nahrání firmwaru do jiného slotu, než pro který byl původně určen. Zároveň by se v jednom ze slotů pořád vyskytovala novější verze, která by byla spuštěna. Testování aktualizací během aktualizace zavaděče MCUboot tedy nefunguje.

V režimu Direct-XIP a RAM load je možné aplikaci zavést přímo ze sekundárního slotu. Vhledem k tomu, že sekundární slot je sdílený mezi aplikací a zavaděčem, není možné tento režim použít.

V operačním systému se sloty pro jednotlivé aplikace konfigurují staticky pomocí maker. Zde nastává problém, nelze definovat primární slot pro zavaděč MCUboot, protože se mění. Kvůli tomu nelze korektně definovat sloty pro MCUboot a proto je aktualizace řešena pomocí sdíleného slotu. Aplikace má informaci pouze o primárním a sekundárním slotu aplikace, s tím že do sekundárního slotu je možné nahrát jak aktualizaci aplikace, tak také aktualizaci zavaděče MCUboot.

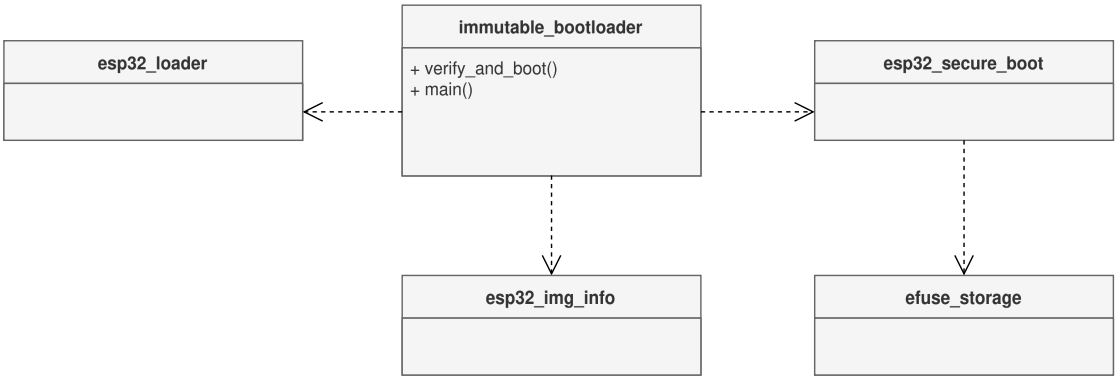
Kapitola 6
Realizace

V této kapitole je popsaná implementace prvního zavaděče a úpravy nezbytné k umožnění aktualizací druhého zavaděče MCUboot. Téměř všechna implementace se nachází v modulu `immutable_bootloader_support`. Kromě toho bylo také nutné provést úpravy v operačním systému Zephyr, projektu MCUboot a modulu `hal_espressif`.

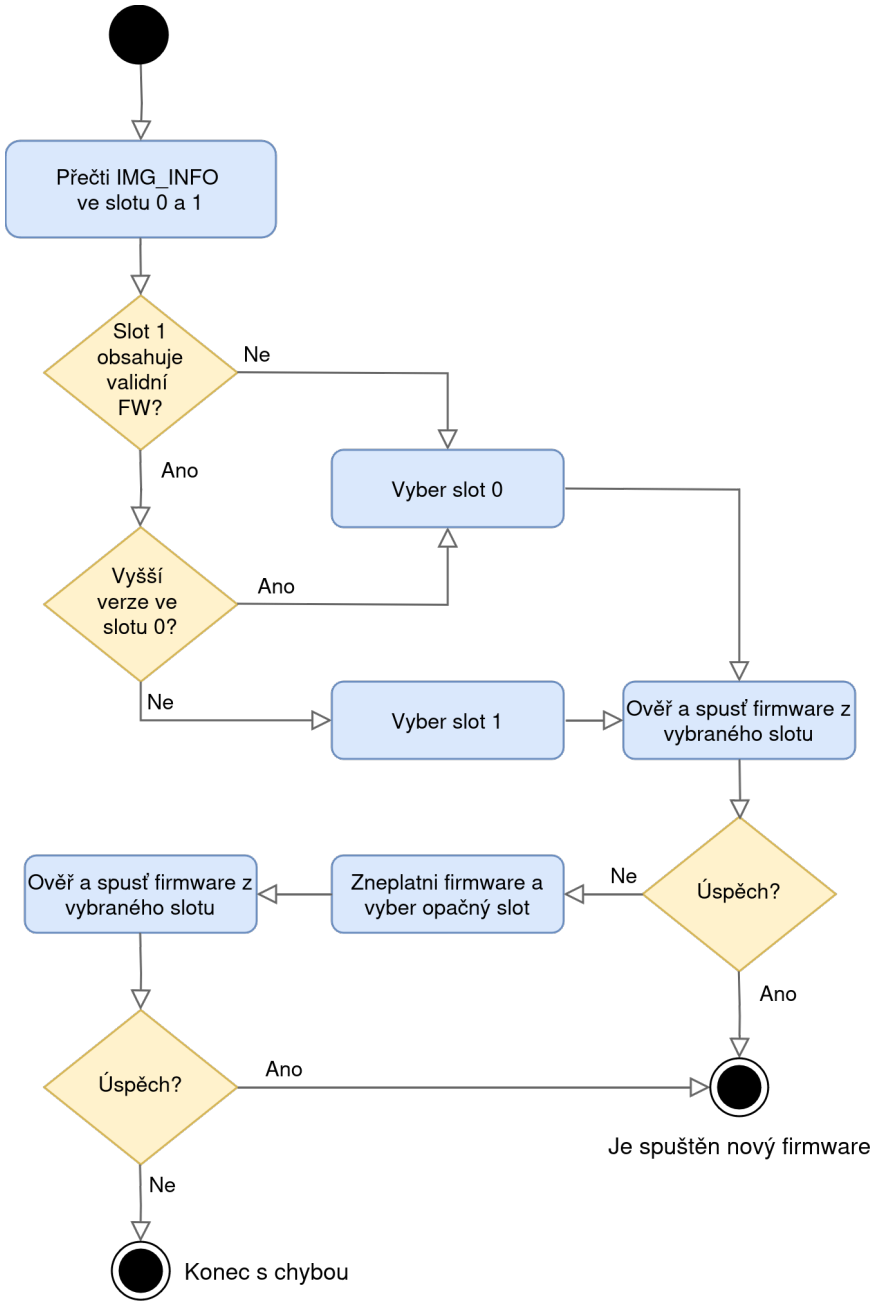
6.1 První zavaděč

První zavaděč dovoluje bezpečné aktualizace druhého zavaděče. Implementace je rozdělena do několika komponent (obrázek 6.1), detailnější popis komponent je uveden v následujících sekcích. Důvodem rozdělení je možnost samostatného otestování jednotlivých komponent.

Jde o jednoduchý zavaděč, který mezi dvěma firmwary vybírá, který spustí. Každý firmware je uložený ve vlastním slotu. Zavaděč se na základě verze rozhodne, jestli spustí firmware ze slotu 0 nebo ze slotu 1. Pokud z vybraného slotu nelze firmware spustit, spustí se firmware ve druhém slotu. Princip jakým zavaděč funguje je popsán na diagramu 6.2. Symbol `CONFIG_BOOT_IMMUTABLE_ESP32` indikuje, že se jedná o tento první zavaděč. Naproti tomu symbol `CONFIG_BOOTLOADER_IMMUTABLE_ESP32` indikuje, že aktuálně sestavovaný firmware je zaváděn tímto prvním zavaděčem.



Obrázek 6.1 Diagram zaváděcí komponenty



Obrázek 6.2 Princip fungování prvního zavaděče

6.1.1 Ověření a spuštění firmware

Pro vybraný slot je spuštěna funkce `veryfy_and_boot`. Funkce firmware ověří a spustí. Ověření firmwaru obnáší kontrolu, zda již není firmware zneplatněný, a následné ověření podpisu firmware. Poté je firmware spuštěn. Spouštění firmwaru řeší komponenta pro zavádění.

Pokud firmware nelze spustit, je zneplatněn, výjimkou je chyba I/O operace. Teoreticky může nastat situace, kdy dočasně nelze číst flash paměť. Ověření firmware by kvůli tomu skončilo chybou, ale zneplatnění firmwaru by proběhlo úspěšně. Tato situace by vedla na zneplatnění validního firmwaru a mohla by znefunkčnit zařízení. Firmware není invalidován pokud nastane chyba čtení flash paměti. Místo toho dojde k resetování procesoru.

6.1.2 Inicializace hardwaru

Zavaděč MCUboot standardně provádí počáteční inicializaci hardwaru. Nyní je MCUboot použitý jako druhý zavaděč. Inicializace hardwaru se přesouvá do prvního zavaděče. Počáteční inicializace zahrnuje následující kroky:

- Aktivuje se super watchdog.
- Nastaví se MPU, dojde k zakázání přístupu do prázdných míst adresního prostoru.
- Resetuje se MMU.
- Nastaví se frekvence procesoru na 80 MHz.
- Inicializuje se SPI flash paměť.
- Aktivuje se watchdog.

Aby bylo možné provést prvotní inicializaci, bylo nutné upravit modul `hal_espressif`. Standardně jsou zdrojové kódy potřebné pro inicializaci hardwaru dostupné pouze pokud je sestavovaný firmware zavaděče MCUboot. Bylo nutné upravit CMake skripty a zahrnout tyto soubory do build systému pokud je sestavován první zavaděč.

Po startu prvního zavaděče není inicializované MMU a flash paměť. Firmware prvního zavaděče běží kompletně v paměti RAM. Pro linkování firmware byl vytvořen nový linker skript `ram_only.ld`, který vychází ze skriptu pro zavaděč MCUboot `mcuboot.ld`.

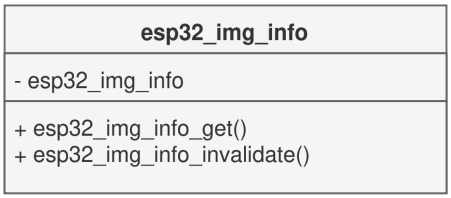
6.2 Struktura popisující firmware

Komponenta `esp32_img_info` řeší umístění struktury `img_info` do firmwaru a manipulaci s ní. Komponenta obsahuje dvě části, první poskytuje rozhraní pro práci se strukturou (její přečtení a invalidace). Druhá vytvoří strukturu `img_info` a zabuduje ji do firmware. První část je možné aktivovat pomocí konfiguračního symbolu `CONFIG_DT_ESP32_IMG_INFO_API`, druhou pomocí `CONFIG_DT_ESP32_IMG_INFO`. Ne každý firmware potřebuje mít strukturu zabudovanou, proto je komponenta rozdělena na dvě části. Například první zavaděč strukturu nemusí obsahovat. Komponenta je znázorněna na diagramu 6.3.

6.2.1 Rozhraní pro přístup k popisu firmware

Komponenta poskytuje rozhraní pro čtení struktury `img_info` z flash paměti a invalidaci firmware. Funkce pro čtení, přečte odpovídající místo flash paměti a zkontroluje magické slovo. V případě, že není slovo validní končí funkce chybou.

Zavaděč před zavedením kontroluje, zda je firmware validní. Validní značka musí být nastavena na hodnotu `0xA4C2FF75`, jinak není firmware validní. Pokud firmware nelze zavést je



Obrázek 6.3 Diagram komponenty `esp32_img_info`

zavaděčem invalidován. Během invalidace se do flash paměti, na místo kde se nachází značka validity, zapíše hodnota `0xFFFF0000`, cílem je aby se po invalidaci změnila její hodnota. Bity ve flash paměti lze bez vymazání sektoru měnit pouze jedním směrem, z nuly na jedna nebo opačně, záleží zda jsou bity sektoru po jeho odstranění nastaveny na hodnotu jedna nebo nula. Zapsáním hodnoty `0xFFFF0000` je jisté, že vždy dojde ke změně hodnoty, aniž by bylo nutné vymazat celý sektor.

6.2.2 Umístění do firmware

Pokud je aktivní konfigurační symbol `CONFIG_DT_ESP32_IMG_INFO` je do firmware zakomponována struktura `img_info`. Hodnota verze, jaká se do firmware zapíše, je udávána konfiguračním symbolem `CONFIG_DT_ESP32_IMG_INFO_VERSION`. Verzi je možné měnit pomocí Kconfig. Na výpisu kódu 6.1 je uvedeno vytvoření struktury. Struktura je umístěna do sekce `.esp32_img_info`. Velikost obrazu je definována proměnou `_image_size`, která je definovaná linker skriptem.

Výpis kódu 6.1 Definice struktury `img_info`

```
#define ATTR __attribute__((__section__(".esp32_img_info"), __used__))

extern const uint32_t _image_size[];

static const ATTR dt_esp32_img_info_t esp32_img_info = {
    .magic          = 0x9DA580F7,
    .version        = 1,
    .size           = sizeof(dt_esp32_img_info_t),
    .img_version    = CONFIG_DT_ESP32_IMG_INFO_VERSION,
    .img_size       = (uint32_t)_image_size,
    .img_flash_addr = IMAGE_FLASH_ADDR,
    .valid          = 0xA4C2FF75,
};
```

Pro správné umístění struktury a spočítání velikosti firmwaru bylo potřeba upravit linker skript. Mezi metadaty a samotnými daty firmwaru byl vytvořen nový paměťový region (výpis kódu 6.2). Do tohoto regionu jsou linkovány data patřící do sekce `.esp32_img_info` (výpis kódu 6.3). Velikost firmwaru se spočítá jako rozdíl adres první sekce `metadata` a poslední sekce, která se linkuje do flash paměti.

Výpis kódu 6.2 Vytvoření paměťového regionu pro strukturu `img_info`

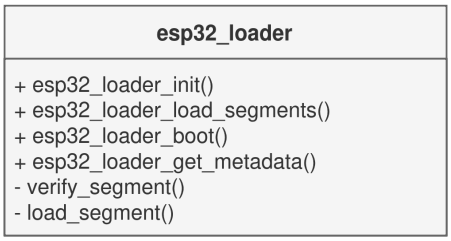
```
MEMORY
{
  metadata (RX): org = METADATA_ORG, len = METADATA_SIZE
#if CONFIG_DT_ESP32_IMG_INFO
  img_info (RX): org = IMG_INFO_ORG, len = ESP32_IMG_INFO_SIZE
#endif
  ROM      (RX): org = ROM_ORG,      len = ROM_SIZE
  ...
}
```

Výpis kódu 6.3 Umístění struktury do vyhrazeného paměťového regionu

```
SECTIONS
{
  ...
#if CONFIG_DT_ESP32_IMG_INFO
  .img_info :
  {
    KEEP(*(.esp32_img_info .esp32_img_info.*))
  } > img_info
#endif
  ...
}
```

6.3 Komponenta pro zavádění firmware

Pro zavádění nového firmware je vytvořena komponenta `esp32_loader`. Zavádění firmware je rozděleno do třech částí: získání informací o firmware z flash paměti, nahrání firmwaru do paměti a jeho spuštění. Zaváděcí komponenta nijak nekontroluje integritu zaváděného firmware. V produkčním prostředí se bude zavaděč používat vždy s aktivovaným secure boot. Secure boot plní zároveň také funkcionalitu kontroly integrity. Struktura komponenty je vyobrazena na diagramu 6.4.



Obrázek 6.4 Diagram zaváděcí komponenty

6.3.1 Přečtení informací o firmware

V této fázi jsou přečteny informace potřebné k zavadění a spuštěni dalšího firmware. Dojde k přečtení metadat, nacházejících se v zaváděném firmware. Po úspěšném přečtení obsahu flash paměti je zkontrolováno magické slovo. Pokud nesouhlasí magické slovo, ve flash paměti se nenachází firmware v podporovaném formátu a tato fáze končí chybou.

6.3.2 Nahrávání do paměti

Další fází je nahrání paměťových segmentu firmwaru do paměti RAM. Segmenty jsou popsány v metadatech. Ke každému segmentu existuje informace o cílové adrese v paměti RAM, jeho velikosti a adrese ve flash paměti, která určuje kde segment začíná. Před nakopírování segmentu do paměti RAM je zkontrolováno, zda během nahrávání obsahu do paměti nedojde k přepsání části aktuálně běžícího firmware. Zaváděcí komponenta neřeší mapování segmentů firmware do adresního prostoru. Namapování se provede po startu firmwaru.

6.3.3 Spuštění nového firmware

Spuštění firmware je finální fází zaváděcí komponenty. V metadatech se také nachází adresa vstupního bodu. Provede se skok na tuto adresu. Vzhledem k tomu, že nový firmware ještě nemá namapovanou flash paměť, musí být vstupní bod umístěn v paměti RAM. Před skokem do nového firmwaru dojde k vypnutí přerušení. Aplikace pro platformu ESP32 standardně předpokládá, že již byla provedena základní konfigurace hardwaru, proto před jejím spuštěním nedochází k de-inicializaci hardwarových prostředků. Nový firmware je spuštěn s již nakonfigurovanou SPI pamětí. Ihned po spuštění nového firmware je nastaven stack pointer, opět se vypnou přerušení a nastaví se adresa nové tabulky vektorů přerušení. Následně dojde k namapování flash paměti do adresního prostoru a startu operačního systému Zephyr. Během startu operačního systému je resetována většina periferií.

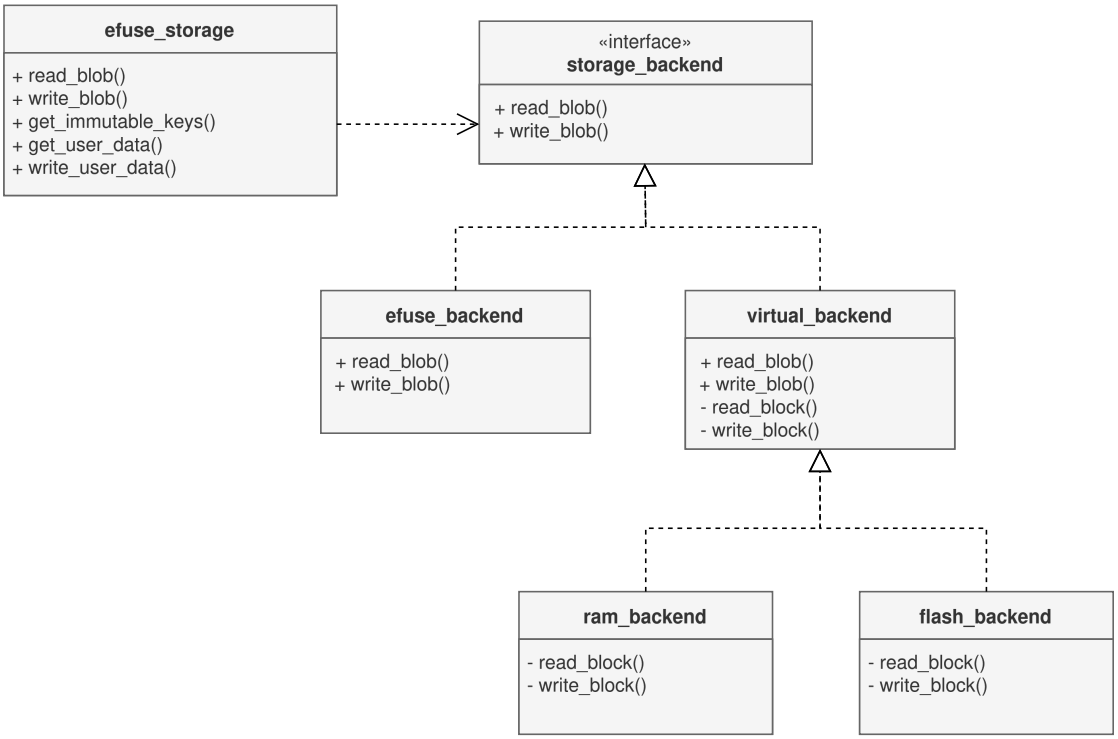
6.4 Přístup k eFuse

Paměť eFuse je programovatelná ROM paměť, jakmile je bit změněn z nuly na jedna, není možné jej žádným způsobem změnit zpět na nulu. Modul `hal_espressif`, který poskytuje podporu pro ESP32 v operačním systému Zephyr, obsahuje rozhraní pro čtení a zápis dat do eFuse. Rozhraní umožňuje operace už po jednom bitu. Čtená nebo zapisovaná data nemusí být zarovnaná. V ESP-IDF SDK je také podpora pro virtuální eFuse v paměti flash nebo RAM, tato funkcionalita v operačním systému Zephyr bohužel chybí.

Vzhledem k tomu, že zápis dat do eFuse je jednorázová operace, není možné pro vyhodnocení a otestování funkčnosti zavaděče zapisovat data přímo do eFuse. Z tohoto důvodu bylo nutné napsat komponentu `efuse_storage`, pomocí které je řešen přístup do eFuse. Diagram komponenty je na obrázku 6.5. Komponenta umožňuje přistupovat k fyzickým eFuse. Je možné ji nakonfigurovat aby byl přístup do eFuse emulovaný pomocí paměti flash případně RAM. Toho je docíleno rozdělením logiky přístupu do paměti na více backendů. Implementován je backend pro přístup k fyzickým eFuse (`efuse_backend`) a backend pro emulaci pomocí paměti flash nebo RAM (`ram_backend` a `flash_backend`). Pří použití virtuální eFuse je možné při inicializaci nahrát do virtuálních eFuse (v paměti flash nebo RAM) obsah fyzických eFuse. Uživatel má také možnost inicializovat virtuální eFuse do určitého stavu, při inicializaci komponenty může být nastaven bit pro aktivaci secure boot a nahrán klíč pro ověření firmware. Vše virtuálně. Díky tomu je možné ověřit implementaci secure boot aniž by bylo potřeba zapisovat data od fyzické paměti.

Rozhraní funkcí `read_blob` a `write_blob` je kompatibilní z rozhraním v `hal_espressif`. Pokud je komponenta nakonfigurovaná pro přístup k fyzickým eFuse, funkce `read_blob` a `write_blob` fungují jako obálka nad funkcemi `esp_efuse_write_field_blob` a `esp_efuse_write_field_blob` z modulu `hal_espressif`.

Kromě obecného přístupu do eFuse komponenta implementuje funkce pro přístup ke klíčům a uživatelským datům. Pomocí Kconfig jde nakonfigurovat, ve kterých blocích se klíče nacházejí. Uživatelská data mohou být použita například pro uložení sériového čísla produktu, případně jakýchkoliv jiných dat. Jde o spojitý datový prostor, který má velikost minimálně jednoho bloku.



Obrázek 6.5 Diagram komponenty pro přístup k eFuse

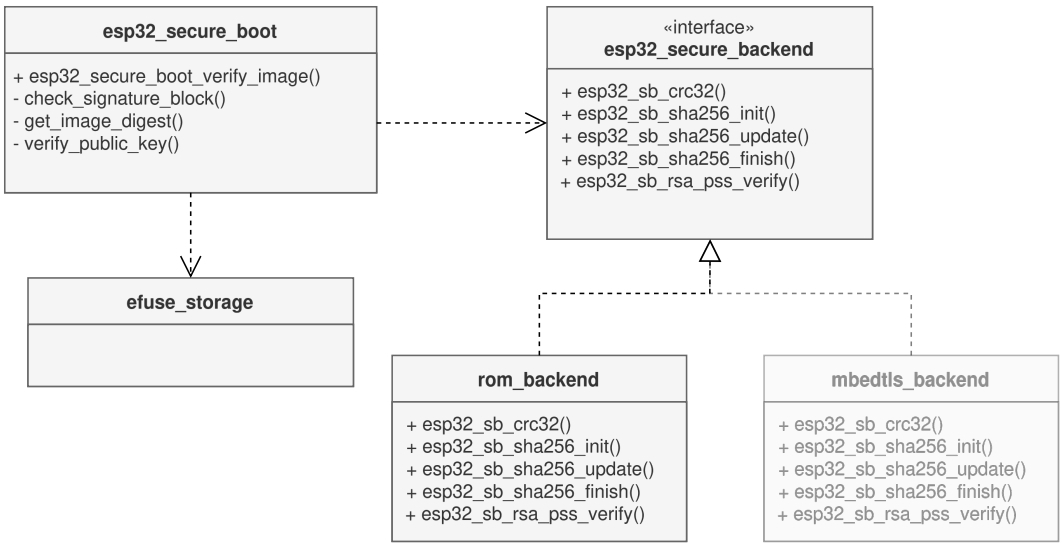
Pomocí Kconfig je možné konfigurovat jaké bloky eFuse tvoří prostor pro uživatelská data. Jednotlivé bloky na sebe nemusí navazovat. Může být vytvořen prostor pro uživatelská data, který se nachází v blocích 3, 7, 8. Tento blok by se skládal ze tří bloků, výsledná velikost tedy bude 768 bitů, respektive 96 bytů. Z uživatelského pohledu je prostor spojitý.

6.5 Secure boot

Komponenta `esp32_secure_boot` řeší ověřování důvěryhodnosti zaváděného firmwaru. Komponenta je rozdělena na dvě části, jedna řeší logiku ověření firmwaru, druhou je kryptografický backend. Toto rozdělení bylo zvoleno s ohledem na testovatelnost jednotlivých komponent. Pokud je komponenta puštěna přímo na některém z čipů ESP32, využívá se backend, který je implementovaný pomocí kryptografických funkcí nacházejících se v ROM paměti čipu. V testovacím prostředí tyto funkce nemusí být dostupné. Kromě testů přímo na čipu je možné spouštět testy například na cloudu v rámci continuous integration. Rozdělení na více backendů umožňuje implementaci kryptografických funkcí jiným způsobem a umožňuje tak otestovat logiku ověření firmwaru. Takový backend může být implementován například pomocí knihovny Mbed TLS. Pro čtení klíčů, které jsou uloženy v paměti eFuse, je využita komponenta `efuse_storage`. Diagram komponenty je na obrázku 6.6.

Ověření firmwaru probíhá v několika krocích. Pokud se firmware nepodaří ověřit je invalidován. Pří dalším spuštění čipu se zavaděč nebude pokoušet zavést nevalidní firmware a pokusí se rovnou zavést firmware v druhém slotu. Kroky během ověřování podpisu firmware jsou následující:

1. Nejprve se zkontroluje, zda je secure boot aktivován nastaveným bitem `SECURE_BOOT_EN` v eFuse. Pokud není aktivovaný, je ověření přeskočeno.
2. Následně se přečte blok s podpisem nacházejícím se na konci firmwaru.
3. Zkontroluje se magické slovo podpisu a ověří jeho kontrolní součet.
4. Proběhne kontrola důvěryhodnosti veřejného klíče, spočítá se hash veřejného klíče a porovná se s hash uloženým v eFuse.
5. Spočítá se hash firmwaru a porovná se s hash uloženým v podpisovém bloku.
6. Ověří se podpis firmwaru.



Obrázek 6.6 Diagram komponenty řešící secure boot

6.6 Definice paměťových regionů

Každý z firmwarů (první zavaděč, MCUboot a aplikace) má vyhrazenou vlastní oblast paměti RAM a flash paměti, která mu náleží. Na výpisu kódu 6.4 je uvedeno rozložení paměti RAM a na výpisu 6.5 rozložení flash paměti. Každý firmware má nastavený devicetree parametr `zephyr,sram` a `zephyr,code-partition`. Tyto parametry udávají jakou část paměti RAM a flash firmware využívá. Toho je využito při linkování firmware a na ESP32 také pro mapování flash paměti do adresního prostoru.

Výpis kódu 6.4 Devicetree definice paměti RAM

```
sram_immutable: mem_region@3fca0000 {
    compatible = "mmio-sram";
    reg = <0x3fca0000 0x14000>;
};

sram_mcuboot: mem_region@3fccb000 {
    compatible = "mmio-sram";
    reg = <0x3fccb000 0x11000>;
};

sram_app: mem_region@3fc80000 {
    compatible = "mmio-sram";
    reg = <0x3fc80000 0x4b000>;
};
```

Výpis kódu 6.5 Devicetree definice flash paměti

```
/* Reserve 128kB for fist bootloader */
boot_partition: partition@0 {
    reg = <0x00000000 0x00020000>;
};

/* Reserve 128kB for slot 0 of mcuboot */
s0: partition@20000 {
    reg = <0x00020000 0x00020000>;
};

/* Reserve 128kB for slot 1 of mcuboot */
s1: partition@40000 {
    reg = <0x00040000 0x00020000>;
};

/* Reserve 1024kB for application primary slot */
slot0_partition: partition@100000 {
    reg = <0x00100000 0x00100000>;
};

/* Reserve 1024kB for shared secondary slot */
slot1_partition: partition@200000 {
    reg = <0x00200000 0x00100000>;
};

/* Reserve 256kB for scratch partition */
scratch_partition: partition@320000 {
    reg = <0x00300000 0x00040000>;
};
```

6.7 Vstupní bod firmwaru

Výchozím vstupním bodem pro platformu ESP32 v operačním systému Zephyr je funkce `__start`, která nejprve provede namapování flash paměti. Následně se pokračuje na funkci `__esp_platform_start`, která provede dodatečnou inicializaci. Poté je spuštěna funkce `z_cstart`, která spustí operační systém Zephyr.

Během inicializace firmware byli objeveny zásadní nedostatky. Během startu vůbec nedochází ke korektnímu nastavení stack pointeru. Při startu se nekorektně vypínají přerušení. K vypnutí přerušení dochází příliš pozdě, dokonce až po nastavení tabulky vektorů přerušení. Tyto nedostatky se standardně neprojevují. Problém nastane pokud firmware, který běží před aplikací má aktivované vícevláknové zpracování nebo jiným způsobem zapnuté přerušení.

V operačním systému Zephyr se na architekturách ARM a RISC-V standardně pro inicializaci systému používá stejný zásobník jako pro obsluhu přerušení. V době inicializace jsou přerušení vypnutá. Pokud je aktivován konfigurační symbol `CONFIG_INIT_STACKS` jsou všechny byty zásobníků nastaveny na hodnotu `0xAA`.

Řešením zmíněných problémů je vytvoření funkce, která bude použita jako vstupní bod. Funkce musí být napsaná v jazyku assembly, protože zatím není nastavený stack pointer. Nejprve dojde k vypnutí přerušení. Pokud je potřeba, inicializuje se zásobník. Nastaví se stack pointer a skočí se na původní vstupní funkci `__start`. Na výpisu kódu 6.6 je uvedena implementace pro čip ESP32-C3 s architekturou RISC-V. Stejnou funkci je možné použít pro jakýkoliv čip z rodiny ESP32 s architekturou RISC-V.

Implementace pro architekturu Xtensa se liší. Pro inicializaci systému se na architektuře Xtensa nepoužívá zásobník pro obsluhu přerušení. Typicky je stack pointer nastaven na konec paměťové oblasti, která je vyhrazená pro daný firmware.

Výpis kódu 6.6 Nová vstupní funkce pro ESP32 architektury RISC-V

```
SECTION_FUNC(reset, __esp32c3_initialize)
    /* Disable interrupts */
    li t0, 0x00000008
    csrc mstatus, t0

#ifdef CONFIG_INIT_STACKS
    /* Pre-populate all bytes in z_interrupt_stacks with 0xAA */
    la t0, z_interrupt_stacks
    li t1, __z_interrupt_stack_SIZEOF
    add t1, t1, t0

    /* Populate z_interrupt_stacks with 0xaaaaaaaa */
    li t2, 0xaaaaaaaa
aa_loop:
    sw t2, 0x00(t0)
    addi t0, t0, 4
    blt t0, t1, aa_loop
#endif

    /*
     * Initially, setup stack pointer to
     * z_interrupt_stacks + __z_interrupt_stack_SIZEOF
     */
    la sp, z_interrupt_stacks
    li t0, __z_interrupt_stack_SIZEOF
    add sp, sp, t0

    call __start
```

6.8 Watchdog

Během celého procesu startování aplikace je aktivní watchdog. K zapnutí dochází již v ROM zavaděči. První zavaděč jej následně resetuje a znovu inicializuje, stejně tak druhý zavaděč MCUboot. Pokud MCUboot provádí aktualizaci firmware musí provést operaci `feed`, jinak by mohlo dojít k resetování systému během aktualizace. Vypnutí watchdogu je provedeno až při inicializaci aplikace. Celý proces bootování od startu ROM zavaděče, až po start aplikace je chráněn proti zamrznutí systému.

Při startu prvního zavaděče je také aktivován super watchdog. Super watchdog resetuje systém, pokud přestane odpovídat na přerušení. Super watchdog funguje plně autonomně. Před vypršením času generuje přerušení a při zachycení přerušení je watchdog automaticky resetován.

6.9 Změny v MCUboot

Společně s použitím jednoho sdíleného sekundárního slotu pro aplikaci a zavaděč bylo potřeba udělat jisté úpravy v projektu MCUboot. Také bylo potřeba provést změny v operačním systému, aby se firmware zavaděče MCUboot při použití jako druhého zavaděče korektně inicializoval.

6.9.1 MCUboot jako druhý zavaděč

Standardně je zavaděč MCUboot používán jako první zavaděč. Provádí počáteční hardwarovou inicializaci a běží kompletně z paměti RAM a oproti aplikaci je linkován pomocí odlišného linker skriptu. Paměť je rozdělena na tři regiony `iram`, `iram_loader`, `dram`. Při takovém rozdělení může aplikace využívat více paměti RAM pro statická data. MCUboot je linkován, tak, že při zavádění aplikace může být `iram` region přepsán. Standardně není dovoleno použití zavaděče MCUboot s podporou vícevláknového zpracování. Důvodem je zmíněné rozdělení do tří regionů. Kód pro zavadění se nachází v regionu `iram_loader`, ostatní instrukce v `iram`. V případě vícevláknového zpracování však může dojít ke změně kontextu na část kódu nacházející se v regionu `iram`. Pokud již byla zavedena aplikace, není možné zaručit, že tento region již nebyl přepsán.

Sériový protokol, který je ve firmě, kde autor pracuje, využíván pro recovery vyžaduje aktivní vícevláknové zpracování. Je tedy nutné aby ho podporoval také MCUboot. V kombinaci s prvním zavaděčem je MCUboot použit jako druhý zavaděč. Již byla provedena počáteční inicializace hardwaru a je nakonfigurované rozhraní SPI spolu s flash paměti. MCUboot je tedy sestaven jako by byl aplikací, flash paměť je po startu mapovaná do adresního prostoru. Je používán výchozí linker skript. žádná část paměti zavaděče MCUboot tedy nemůže být přepsána. Zároveň je však většina instrukcí a část dat umístěna ve flash paměti, takže zavaděč zabírá v paměti RAM méně místa.

6.9.2 Varianta pro slot 1

Standardně je sestavena varianta pro slot 0. Pomocí Kconfig lze zvolit jakou variantu sestavit. Nastavením konfiguračního symbolu `CONFIG_BOOTLOADER_IMMUTABLE_ESP32_S1_VARIANT` bude výsledný firmware sestaven pro slot 1. Podle cílového slotu je nastaven primární slot pro aktualizaci zavaděče a při inicializaci firmwaru je do adresního prostoru mapována správná oblast flash paměti.

6.9.3 Definice slotů

Jednotlivé sloty, se kterými zavaděč MCUboot pracuje se definují v souboru `sysflash.h` projektu MCUboot. Pro každý aktualizovatelný firmware je definován pár slotů (primární a sekundární).

Sloty jsou definovány pomocí makra `FLASH_AREA_IMAGE_<X>_SLOTS`. Aplikace je firmware 0, jako její primární slot je definována oblast flash paměti `slot0_partition`, jako sekundární `slot1_partition`. Zavaděč MCUboot je firmware 1. Sekundární slot je sdílený s aplikaci, primární slot závisí na variantě MCUboot. Vždy je to opačný slot, než pro který je zavaděč sestaven. Zavaděč ve slotu 0 má primární slot nastavený na oblast `s1`, druhá varianta naopak. Definice je zobrazena na ukázce 6.7.

Výpis kódu 6.7 Definice slotů pro zavaděč MCUboot

```
#define FLASH_AREA_IMAGE_0_SLOTS slot0_partition, slot1_partition

#if CONFIG_BOOTLOADER_IMMUTABLE_ESP32_S1_VARIANT
    #define FLASH_AREA_IMAGE_1_SLOTS s0, slot1_partition
#else
    #define FLASH_AREA_IMAGE_1_SLOTS s1, slot1_partition
#endif
```

6.9.4 Podpora sdíleného sekundárního slotu

Zavaděč MCUboot standardně nepodporuje sdílení sekundárního slotu. Po spuštění MCUboot zjišťuje, zda není v sekundárním slotu dostupná aktualizace. Postupně iteruje přes všechny aktualizované aplikace, a zjišťuje zda má provést výměnu primárního a sekundárního slotu. V případě sdíleného sekundárního slotu je nutné kontrolovat, zda firmware v sekundárním slotu je určený do právě vybraného primárního slotu. Tato úprava byla převzata z NRF Connect SDK a upravena, aby fungovala s implementací pro ESP32. Obraz firmware pro architekturu ARM obsahuje na začátku adresu reset vektoru, na kterou se skočí [19]. Tato adresa je adresou do flash paměti. Adresa reset vektoru tedy musí být uvnitř primárního slotu aby mohla být provedena aktualizace. Na platformě ESP32 se v metadatech nachází vstupní adresa do firmware, jde ale o adresu do paměti RAM. Nelze ji použít k rozeznání, do kterého slotu firmware patří. Slot, do kterého firmware patří se zjistí ze struktury `img_info`, která obsahuje adresu flash paměti, na kterou firmware patří.

6.10 Informace o firmware

Operační systém Zephyr poskytuje shell subsystém. Díky tomu je možné se zařízením komunikovat pomocí textového uživatelského rozhraní. V rámci práce je vytvořený příkaz `dt_fwinfo`. Příkaz zobrazí hash aktuálně běžící aplikace, aktuální verzi zavaděče MCUboot a slot, ze kterého je spouštěn. Příkaz je používán během testování aktualizací k vyhodnocení, zda aktualizace proběhla úspěšně. Příkaz lze zároveň využít před aktualizací zavaděče. Při aktualizaci zavaděče je třeba vědět jakou variantu zavaděče MCUboot do zařízení nahrát, jestli variantu pro slot 0 nebo 1. Pomocí příkazu lze zjistit aktuální aktivní slot. Do zařízení má být nahrána varianta pro opačný slot. Informace o aktuální verzi a aktivním slotu zavaděče MCUboot je vyhodnocena na základě inspekce struktury `img_info` v obou slotech, ve kterých se zavaděč může nacházet. Aktivní slot zavaděče MCUboot je ten, ve kterém se nachází novější verze a zároveň je validní.

Výpis kódu 6.8 Příkaz dt_fwinfo

```
uart:~$ dt_fwinfo
firmware hash: b74f076ad90283056b04ce5bd5ab4ec5d7eb86adbb69564a76b
mcuboot version: 25
mcuboot slot: 0
```

6.11 Integrace do build systému

Výsledná implementace byla integrována do build systému operačního systému Zephyr. Bylo využito rozšíření sysbuild, které umožňuje sestavit zároveň několik aplikací. Výstupem sestavení je aplikace, obě varianty zavaděče MCUboot, firmware prvního zavaděče a jeden binární soubor, který kombinuje všechny firmwary do jednoho souboru. Výsledné binární soubory jsou zároveň dle konfigurace také podepsané a je možné je přímo nahrát do zařízení. K nasazení na produkční prostředí zbývá jen zapsat potřebná data do paměti eFuse.

6.11.1 Sestavení zavaděčů

Pří použití rozšíření sysbuild je automaticky sestavena aplikace společně se zavaděčem MCUboot. Funkcionalitu je možné pomocí CMake rozšiřovat. Funkce `ExternalZephyrProject_Add` přidá do build systému další aplikaci. Tímto způsobem je do build systému zaraženo sestavení firmwaru pro první zavaděč a ještě jeden MCUboot jako varianta pro slot 1. Sestavení zavaděče MCUboot ve variantě pro slot 1 zahrnuje zkopírování konfigurace, která se používá pro sestavení varianty pro slot 0 a následné nastavení konfiguračního symbolu `CONFIG_BOOTLOADER_IMMUTABLE_ESP32_S1_VARIANT`, který indikuje, že jde o firmware pro druhý slot.

6.11.2 Podepisování firmware

Během sestavení aplikace jsou binární soubory podepsány. Zephyr umožňuje nastavení vlastního skriptu pro podepsání firmware. Byl vytvořený skript `esp32_sign.cmake`, který řeší podepsání všech firmware. Aplikace je podepsáná pomocí nástroje `imgtool.py`, klíč je možné nastavit konfiguračním symbolem `CONFIG_MCUBOOT_SIGNATURE_KEY_FILE`. Zavaděč MCUboot je podepsán nejprve nástrojem `espsecure.py`, klíč je specifikován pomocí symbolu `CONFIG_ESPSECURE_SIGNATURE_KEY`. Následně je binární soubor ještě jednou podepsán, jako aplikace, nástrojem `imgtool.py`. První zavaděč je podepsán jen pomocí `espsecure.py`.

Nástroj `espsecure.py` umístí na konec firmwaru podpisový blok. Tento blok je ve výchozím stavu umístěn na adrese zarovnané na 4 KB. Vzhledem k tomu, že firmware zavaděče MCUboot je následně ještě jednou podepsán, nedává zarovnání bloku smysl. Po přidání hlavičky stejně nebude blok zarovnaný. Také se tím zbytečně navyšuje velikost výsledného binárního souboru. Pro účely podepisování zavaděče MCUboot byl skript upraven a byl přidán přepínač `--no-align`, který dovolí zapsání bloku přímo na konec firmwaru.

6.11.3 Nahrávání více firmware do zařízení

Pro nahrávání aplikace se v operačním systému Zephyr standardně využívá nástroj `west flash`. Spolu s rozšířením sysbuild je příkaz spuštěn pro každou aplikaci zařazenou do build systému. Během sestavení aplikace je vytvořen soubor `runners.yaml` obsahující informace o tom, jak firmware nahrát do zařízení. Zde je uvedena adresa na kterou patří. Během sestavení je nastavena korektní adresa, dle rozložení flash paměti, definovaném pomocí devicetree.

6.11.4 Sjednocení do jednoho binárního souboru

V rámci práce byl vytvořen python skript, který pomocí binárních souborů a odpovídajících adres na vstupu sjednotí několik firmware do jednoho binárního souboru. Tento soubor může být použitý pro naprogramování zařízení. Ke sjednocení dojde automaticky poté, co jsou vytvořeny a podepsány binární soubory.

6.12 Modul

Většina implementace se nachází v modulu `immutable_bootloader_support`. Níže je popsaná struktura tohoto modulu:

```
immutable_bootloader_support/
├── bootloader/ ..... zdrojové kódy prvního zavaděče
├── cmake/ ..... cmake skripty
├── include/ ..... veřejné hlavičkové soubory
├── samples/
│   └── app_with_updatable_mcuboot/ ..... vzorový projekt
├── scripts/
│   ├── build/ ..... skripty používané pro sestavení
│   └── pylib/ ..... python skripty používané pro testování
├── subsys/ ..... zdrojový kód jednotlivých komponent
├── sysbuild/ ..... soubory související se sysbuild
├── tests/
│   └── mcumgr_update/ ..... test aktualizací
└── zephyr/
    └── module.yml/ ..... popis tohoto modulu
```

6.13 Postup při portaci na jiné ESP32

První zavaděč je navržen tak, aby podporoval i jiné čipy z rodiny ESP32. Toho je docíleno zejména díky abstrakci ze strany HAL vrstvy `hal_espressif`. Sám autor byl schopný zavaděč spustit na čipu ESP32-S3 a ESP32-C3, které jsou odlišné architektury. Pro podporu dalšího čipu je však nutné provést změny v operačním systému Zephyr. Každý čip má definovaný svůj vlastní inicializační kód a linker skripty. Bohužel je zdrojový kód separátní i pokud jde o procesory stejné architektury. V následujících bodech jsou shrnuty kroky potřebné k přidání podpory pro další čip.

1. Vytvořit linker script pro první zavaděč, tak aby byl firmware spuštěn pouze z paměti RAM.
2. Definovat rozdělení paměti RAM a flash pomocí devicetree.
3. Vytvořit vstupní funkci firmware, která nastavý stack pointer a vypne přerušení.
4. Pro sestavení MCUboot použít aplikační linker skript.
5. Upravit inicializaci v operačním systému, tak aby počáteční hardwarovou inicializaci prováděl první zavaděč.
6. MCUboot po startu namapuje svou část flash paměti, stejně jako aplikace.

Kapitola 7
Vyhodnocení a testování

7.1 Diskuze odolnosti řešení

Cílem této práce bylo vytvoření řešení pro bezpečné aktualizace zavaděče MCUboot odolné vůči chybám z důvodu výpadku napájení nebo korupce data. V průběhy aktualizací mohou nastat jisté hraniční situace, které by mohli vést k neúspěchu v průběhu aktualizace zavaděče. V této sekci jsou popsané možné situace a jak jsou ve finálním řešení zohledněny.

Nahrání aktualizace pro nesprávný slot
Po spuštění firmwaru v nesprávném slotu by po spuštění došlo k namapování nesprávné oblasti flash paměti. V adresním prostoru nebude korektně namapovaný firmware, což povede na nedefinované chování. Může dojít i k znefunkčnení zařízení. Aby se předešlo spuštění firmwaru z nesprávného slotu, první zavaděč kontroluje, zda je firmware umístěný na správné adrese. Adresa, na kterou firmware patří je umístěna ve struktuře `img_info`. Adresu firmwaru zároveň kontroluje také zavaděč MCUboot. V případě, že se do sekundárního slotu nahraje špatný firmware, zavaděč neprovede aktualizaci.

Chyba během čtení flash paměti
Dočasná chyba čtení flash paměti může způsobit falešnou chybu během ověřování firmware. Pokud by k tomu došlo, mohlo by dojít k zneplatnění validního firmwaru. Při detekci chyby čtení flash paměti je zařízení restartováno.

Výpadek napájení během aktualizace
Po výpadku napájení během aktualizace se ve flash paměti nemusí nacházet spustitelný firmware. Z toho důvodu první zavaděč vždy vybírá mezi dvěma firmwary. Po výpadku firmware se nový firmware nepodaří ověřit a bude spuštěn starý firmware. Samotný MCUboot umí detekovat přerušení aktualizace. Po opětovném spuštění zařízení je aktualizace dokončena.

Korupce dat při aktualizaci
Jde o podobný případ případ jako výpadek napájení během aktualizace. Korupce dat může nastat ve dvou případech. Během nahrávání firmwaru do sekundárního slotu a teoreticky také při výměně obsahu slotů. V prvním případě korupci dat odhalí MCUboot a aktualizace je přerušena. V druhém případě korupci dat zjistí první zavaděč. Slot nebude obsahovat validní firmware a dojde ke spuštění původního firmwaru.

Aktualizace na nefunkční MCUboot
Také může nastat situace, kdy je zařízení aktualizováno na novější verzi zavaděče MCUboot, který nefunguje. Může se stát, že po spuštění zařízení havaruje. První zavaděč tento problém nijak neřeší. Před vydáním aktualizace je vždy nutné mít firmware otestovaný. V případě,

že by k takovému případu stejně došlo, je na procesorech rodiny ESP32 dostupný zavaděč v ROM paměti a do zařízení je tak stále možné nahrát korektní firmware.

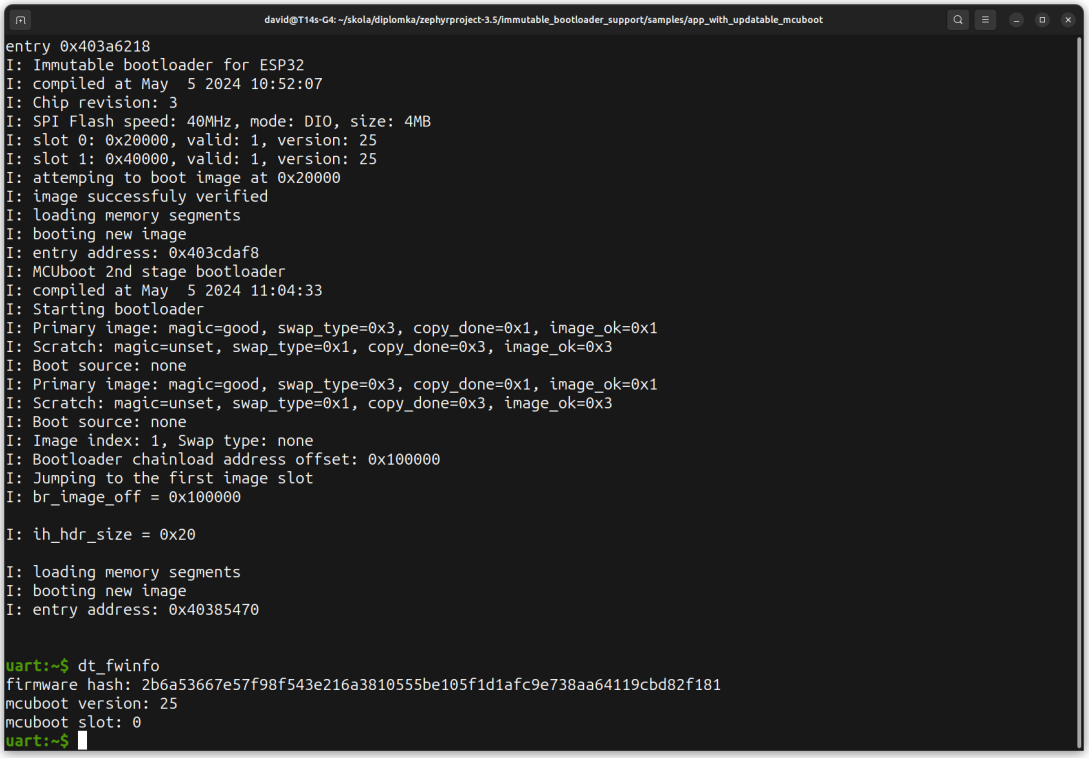
7.2 Vzorový projekt

Součástí práce je vzorový projekt pro demonstraci aktualizace zavaděče MCUboot a aplikace. Projekt se nachází ve složce `samples/app_with_updatable_mcuboot`. Aplikace obsahuje SMP server a textové uživatelské rozhraní. Díky přítomnému SMP serveru je možné aktualizaci firmware provést pomocí nástroje MCUmgr. Společně s aplikací jsou sestaveny obě varianty zavaděče MCUboot a první zavaděč. Virtuálně je také aktivovaný secure boot. K podpisu firmwaru se využívají klíče nacházející se v podsložce `keys`.

Aplikaci je možné sestavit pro čip ESP32-C3, případně také ESP32-S3. Na příkladu 7.1 je uvedeno její sestavení a nahrání na vývojovou desku ESP32-C3-DevKitM-1. Do zařízení je nahrán první zavaděč, obě varianty zavaděče MCUboot a aplikace. Na obrázku 7.1 je možné vidět výpis logů ze zařízení po jeho restartu.

Výpis kódu 7.1 Sestavení a nahrání vzorového projektu

```
$ west build -b esp32c3_devkitm --sysbuild
$ west flash
```



Obrázek 7.1 Výpis logů ze zařízení po spuštění vzorového projektu

Pro provedení aktualizace zavaděče MCUboot je třeba sestavit firmware, který je vyšší verze. Pomocí menuconfig je možné změnit verzi. Nástroj menuconfig je možné spustí příkazem `west build -t mcuboot_menuconfig`. Konkrétně je potřeba nastavit vyšší hodnotu konfiguračnímu symbol `CONIFG_IMG_INFO_VERSION` a následně firmware znovu sestavit pomocí `west build`. Pro

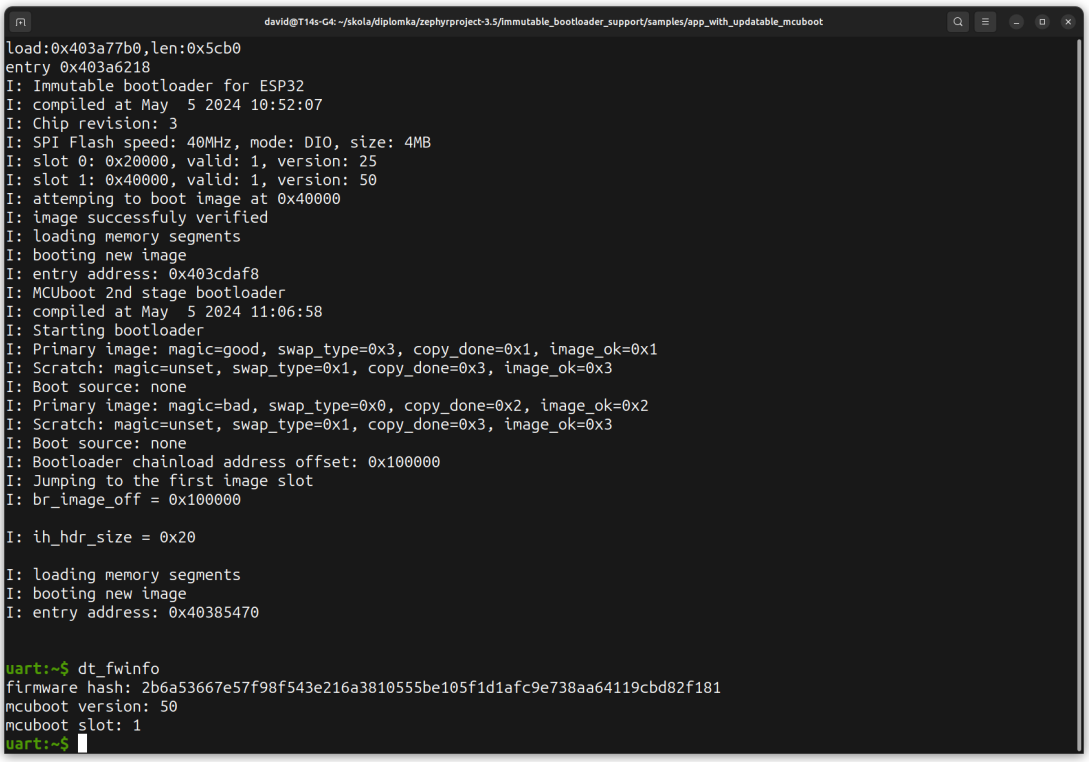
nahrání nové verze zavaděče MCUboot se použije nástroj MCUmgr. Do zařízení je nutné nahrát variantu pro opačný slot, než jaký je nyní aktivní. Aktuálně aktivní slot zavaděče je možné zjistit pomocí textového uživatelského rozhraní operačního systému Zephyr. K tomuto účelu byl implementován příkaz `dt_fwinfo`. Postup nahrání aktualizace do zařízení pomocí MCUmgr je uveden níže. Po nahrání souboru do zařízení je nutné firmware potvrdit. Firmware je identifikován podle jeho hash.

Výpis kódu 7.2 Nahrání aktualizace do zařízení

```
$ mcumgr --conntype serial --connstring='dev=/dev/ttyUSB0,baud=115200' \
    image upload <filename>

$ mcumgr --conntype serial --connstring='dev=/dev/ttyUSB0,baud=115200' \
    image confirm <hash>
```

Po restartu bude zatím stále spuštěna původní verze zavaděč MCUboot. V sekundárním slotu bude detekována připravená aktualizace a dojde k vyměně dat mezi primárním a sekundárním slotem. Jak je možné vidět na obrázku 7.2, po dalším restartu zařízení již bude spuštěna nová verze zavaděče MCUboot. Do zařízení byl nahrán MCUboot verze 50.



Obrázek 7.2 Výpis logů ze zařízení po provedení aktualizace

7.3 Vzorové projekty v Zephyr RTOS

Operační systém Zephyr poskytuje mnoho vzorových projektů. Některé z nich byly otestovány v kombinaci s prvním zavaděčem a zavaděčem MCUboot. Byl vyzkoušen například vzorový projekt pro Wi-Fi konektivitu nebo projekt umožnující aktualizace firmware pomocí bluetooth. Některé

projekty bohužel nefungují. Aplikace se vůbec nespustí. Konkrétně například vzorový projekt pro Wi-Fi konektivitu. Bylo zjištěno, že v kombinaci s MCUboot tyto projekty na operačním systému ve verzi 3.5 standartě nefungují. Chyba je na straně podpory čipů ESP32. Problém způsobují špatně napsané linker skripty. Data patřící do flash paměti jsou nekorektně zarovnaná. Problém byl již dříve někým nahlášen a je řešen. Na aktuální vývojové verzi systému by již měl být vyřešen.

7.4 Využití paměti

V této sekci je uvedeno paměťové využití prvního zavaděče na čipu ESP32-C3 (tabluka 7.1) a ESP32-S3 (tabluka 7.2), uvedeno je využití paměti flash a RAM. Pro srovnání jsou v tabulce také paměťové požadavky zavaděče MCUboot. První zavaděč běží oproti zavaděči MCUboot celý v paměti RAM. Z toho důvodu má větší požadavky na paměť RAM. Velikost potřebné paměti RAM však není problém, paměť, ve které se nachází první zavaděč, je přepsaná aplikací. Výsledný binární soubor, který se nahrává do zařízení neobsahuje sekce označené jako `no load`. Jde o neinicializovaná data nebo nebo data inicializovaná na hodnotu 0. Sem patří například zásobníky nebo halda. Z tohoto důvodu je u prvního zavaděče vyšší požadavek na paměť RAM, než na paměť flash.

Tabulka 7.1 Paměťová náročnost zavaděčů na čipu ESP32-C3

	RAM	**Flash**
první zavaděč	54368 B	34992 B
MCUboot	40156 B	101806 B

Tabulka 7.2 Paměťová náročnost zavaděčů na čipu ESP32-S3

	RAM	**Flash**
první zavaděč	58364 B	43488 B
MCUboot	38484 B	101857 B

7.5 Jednotkové testy

V rámci práce bylo v plánu vytvořit také jednotkové testy a izolovaně otestovat jednotlivé komponenty prvního zavaděče. Bohužel nebyly testy z časových důvodů realizovány. Nicméně jednotlivé komponenty byly navrženy s ohledem na testování a testy je možné v budoucnu dopsat. Testy mohou být pouštěny přímo na čipu, nebo v prostředí, které neumožňuje přístup k hardwaru, například v rámci continuous integration.

7.6 Testování aktualizací

Testování aktualizací je nezbytnou součástí testování. Testování aktualizací testuje systém jako celek. Testování aktualizací probíhalo na fyzických zařízeních. Byly použity vývojové desky ESP32-C3-DevKitM-1 a ESP32-S3-DevKitM-1.

7.6.1 Simulace chyb

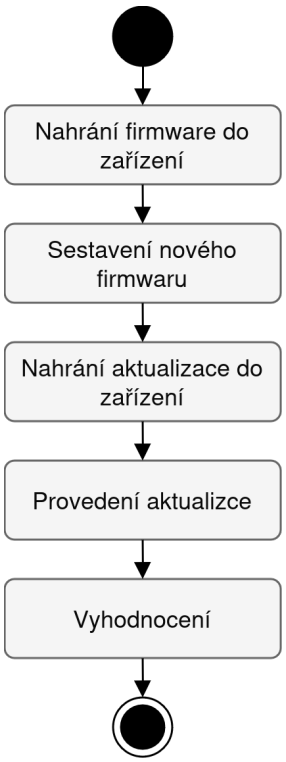
Během provádění aktualizace může dojít k chybám v podobě výpadku napájení, nebo korupce dat. Pro předejití možných problémů byl systém otestován simulováním těchto chyb. Do zařízení

byl nahrán vzorový projekt popsaný v sekci 7.2. Chyba výpadku napájení nebo korupce dat byla simulována vymazáním jednoho sektoru, ve kterém se nachází firmware. Nejprve proběhlo vymazání sektoru začínajícího na adrese `0x21000` a vyhodnocení. Následně byl do zařízení nahrán opět vzorový projekt a došlo k vymazání sektoru druhého slotu na adrese `0x41000`. Tím test pokryl spuštění firmwaru z obou slotů. Na obou vývojových deskách proběhl test úspěšně. Slot z vymazaným sektorem obsahoval nevalidní firmware a došlo ke spuštění firmwaru z druhého slotu.

7.6.2 Testy pomocí pytest

Operační systém Zephyr obsahuje testovací framework Twister. Pomocí něj je možné možné psát jednotkové a integrační testy. Nově je nástroj integrovaný jako doplněk do testovací frameworku pytest a umožňuje psaní komplexnějších testů pomocí jazyka python. Testování aktualizací je napsáno pomocí frameworku pytest. Současně s psaním testů vznikla python knihovna implementující podpůrnou funkcionalitu pro provádění testů. V knihovně jsou implementované komponenty pro sestavení firmwaru, nahrávání firmwaru do zařízení, nahrávání aktualizace do zařízení a vyčítání informací o firmwaru pomocí příkazu `dt_fwinfo`.

Samotný test je rozdělen do několika fází. Nejprve dojde k uvedení zařízení do známého stavu, do zařízení se nahrají potřebné firmwary. Následně se sestaví nový firmware, který je použitý pro aktualizaci. Aktualizace se následně nahraje do zařízení pomocí MCUmgr. Aktualizace se potvrdí a čip se restartuje. Pokud je prováděna aktualizace zavaděče je čip po aktualizaci restartován ještě jednou. Na závěr se přečtou informace o aktuálním stavu firmwaru a vyhodnotí se výsledek testu. Proces je vyobrazen na obrázku 7.3.



Obrázek 7.3 Průběh testu aktualizací

Popsaný test je spuštěn několikrát pro každý testovací případ. Během testování aktualizace zavaděče se testuje několik testovacích případu. Těmi jsou následující:

Aktualizace na novou verzi zavaděče
Testuje se, zda je možné provést aktualizaci zavaděče MCUboot.

Nahrání aktualizace firmwaru pro nesprávný slot
Do zařízení je nahrána varianta MCUboot pro nesprávný slot. Test je úspěšný, pokud aktualizace nebyla provedena.

Aktualizace na nižší verzi zavaděče
Testuje se nahrání nižší verze zavaděče, než je aktuální verze. Test je úspěšný, pokud je spuštěna původní verze.

Nahrání firmware s neplatným podpisem
Do zařízení je nahrán firmware, který je podepsaný nedůvěryhodným klíčem. Test je úspěšný pokud aktualizace neproběhne.

Každý z testů byl proveden dvakrát. Jednou pro MCUboot ve slotu 0 a po druhé pro MCUboot ve slotu 1. Celkem tedy 8 testovacích případů. Vhledem k úpravám provedeným v projektu MCUboot je třeba testovat také aktualizaci aplikace. Během testu aktualizace aplikace byly otestovány dva případy:

Aktualizace na novou verzi aplikace
Testuje se, zda je možné provést aktualizaci.

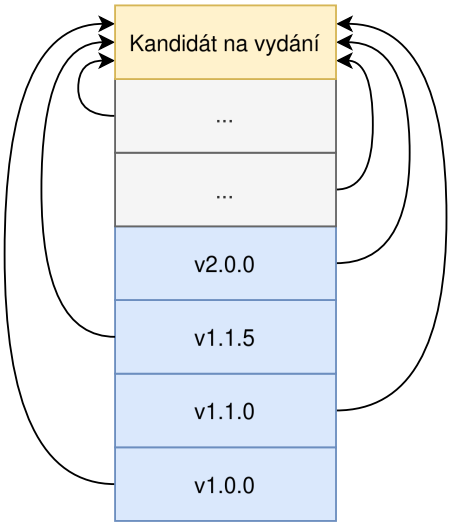
Nahrání firmwaru aplikace s neplatným podpisem
Test je úspěšný pokud není aktualizace provedena.

Na obou vývojových deskách proběhly všechny testy úspěšně.

7.6.3 Testování aktualizací v praxi

V praxi je testování aktualizací nutné rozdělit na dvě části. Je nutné testovat, zda z nově vydané verze firmware bude možné provést aktualizaci na novější verzi. Druhou částí je testování kompatibility se všemi staršími verzemi.

První část, tedy test zda je možné provést aktualizaci na budoucí firmware je popsaná v sekci 7.6.2. Druhá část testu zahrnuje otestování aktualizace ze všech předchozích verzí na verzi aktuální. Pro tuto část testování je možné se inspirovat způsobem na obrázku 7.3. Jednotlivé kroky bude jen potřeba upravit. Nejprve by se do zařízení nahrála starší verze firmwaru, poté by došlo k aktualizaci na aktuálně testovanou verzi. Na závěr se test vyhodnotí. Vyhodnocení by mohlo probíhat stejným způsobem jako je popsáno v sekci 7.6.2. Takto se bude test opakovat, dokud nebude otestována aktualizace ze všech možných verzí. Tento proces je také popsaný na diagramu 7.4.



Obrázek 7.4 Testování kompatibility se starší verzí firmwaru

Kapitola 8
Závěr

Cílem práce bylo přidání podpory pro aktualizace zavaděče MCUboot na platformě ESP32 v rámci operačního systému reálného času Zephyr. V první polovině práce jsou popsané technologie používané v rámci práce. Je zde popsána platforma ESP32, zavaděč MCUboot a operační systém reálného času Zephyr.

Inspirací pro řešení bylo nRF Connect SDK od firmy Nordic Semiconductor, které je postavené na operačním systému Zephyr a jako zavaděč je rovněž podporován MCUboot. Řešení od firmy Nordic Semiconductor umožňuje použiti zavaděče MCUboot jako druhého aktualizovatelného zavaděče. Procesory od firmy Nordic Semiconductor jsou architektury ARM, zatímco platforma ESP32 využívá architektu RISC-V, případně Xtensa. S tím souvisela celá řada odlišnosti, které bylo potřeba vyřešit.

Aktualizace zavaděče MCUboot je umožněna implementováním nového zavaděče a použitím zavaděče MCUboot jako druhého. Pro MCUboot jsou ve flash paměti rezervovány dva sloty, první zavaděč vybírá, který z těchto dvou spustí. Při provádění aktualizace je nový firmware nahraný do opačného slotu, než ze kterého je právě spouštěn. Výběr probíhá na základě verze firmware. Do obrazu firmwaru byla přidána struktura popisující daný firmware. Zde se mimo jiné nachází také informace o jeho verzi. Během aktualizace nikdy nedochází k přepsání aktuální verze zavaděče. Aktualizace je tak možné provést bezpečně, při chybě dojde ke spuštění starého zavaděč.

Implementace prvního zavaděče byla rozdělena do několika komponent, které řeší jednotlivé části problému. Komponenty byly navrženy s ohledem na testovatelnost. K zajištění podpory pro secure boot musela být podpora pro secure boot zajištěna také novým prvním zavaděčem. Jedna z komponent řeší secure boot. Pro ověřování zaváděného firmwaru bylo využito dostupných hardwarových prostředků na čipech ESP32. Klíče jsou uchovávány v jednorázově zapisovatelné paměti eFuse. Pro přístup do eFuse byla vytvořena další komponenta, která umožňuje přístup do eFuse emulovat pomocí paměti RAM, či flash.

Aktualizace zavaděče provádí samotný zavaděč MCUboot, který byl pro tyto účely upraven. Aktualizace zavaděče probíhá stejným způsobem jako aktualizace aplikace, pomocí nástroje MCUmgr. Řešení je integrované do build systému operačního systému. Aplikaci je možné sestavit společně s prvním zavaděčem a oběma variantami zavaděče MCUboot. Výstupem sestavení jsou již podepsané binární soubory, které je možné nahrát do zařízení nebo použít pro aktualizaci firmwaru. Automaticky také dochází ke sjednocení všech binární souborů do jednoho. Tento soubor je možné použít při programování zařízení na produkčním prostředí.

Výsledkem je funkční řešení, které bylo otestováno na čipech ESP32-C3 a ESP32-S3. Otestování výsledného řešení probíhalo pomocí testování aktualizací. Vzhledem k upravenému zavaděči MCUboot nebyly testovány jen aktualizace zavaděče, ale také aplikace. Testování aktualizací otestovalo celý systém jako celek. Testy byly spuštěny přímo na vývojových deskách. Řešení

bylo navrženo s ohledem na testovatelnost jednotlivých komponent, z časových důvodů bohužel nebyly jednotkové testy realizovány.

Během vyhodnocování byly zjištěny nedostatky v podpoře čipů rodiny ESP32 v rámci operačního systému Zephyr. V kombinaci se zavaděčem MCUboot v některých případech operační systém vůbec nefunguje. Po startu aplikace dojde k pádu celého systému. Po vyřešení těchto nedostatků ze strany firmy Espressif Systems bude výsledné řešení nasazené na nových produktech využívající procesory z rodiny ESP32.

Zkratky

ADC Analog-to-digital converter.

AES Advanced Encryption Standard.

AMP Asymmetric multiprocessing.

API Application Programming Interface.

CCTV Closed-circuit television.

CLI Command Line Interface.

DDOS Denial-of-service attack.

DIO Dual Input/Output.

DOUT Dual Output.

DRAM Data RAM.

DROM Data ROM.

ECDSA Elliptic Curve Digital Signature Algorithm.

EDDSA Edwards-curve Digital Signature Algorithm.

ELF Executable and Linkable Format.

GPIO General-purpose input/output.

I2C Inter-Integrated Circuit.

I2C Inter-IC Sound.

IoT Internet of Things.

IP Internet Protocol.

IRAM Instruction RAM.

IROM Instruction ROM.

JTAG Joint Test Action Group.

MAC Media Access Control.

MMU Memory Management Unit.

MPU Memory Protection Unit.

POSIX Portable Operating System Interface.

QIO Quad Input/Output.

QOUT Quad Output.

RAM Random-access memory.

ROM Read-only memory.

RSA Rivest–Shamir–Adleman.

RTC Real-time clock.

RTOS Real-time Operating System.

SDK Software Development Kit.

SMP Simple Management Protocol.

SoC System on Chip.

SPI Serial Peripheral Interface.

TCP Transmission Control Protocol.

TLV Type-Length-Value.

UART Universal asynchronous receiver-transmitter.

UDP User Datagram Protocol.

USB Universal Serial Bus.

Wi-Fi Wireless Fidelity.

XIP Execute in place.

Bibliografie

1. CONSTANTIN, Lucian. *Thousands of hacked CCTV devices used in DDoS attacks*. PC World, 2016. Dostupné také z: https://www.pcworld.com/article/415443/thousands-of-hacked-cctv-devices-used-in-ddos-attacks.html.
2. KLEIDERMACHER, David; KLEIDERMACHER, Mike. Introduction to embedded systems security. In: *Embedded Systems Security*. Elsevier, 2012. ISBN 978-0-12-386886-2. Dostupné z DOI: 10.1016/c2010-0-67275-0.
3. LACAMERA, Daniele. *Embedded systems architecture*. Birmingham, England: Packt Publishing, 2018. ISBN 1788832507.
4. ESPRESSIF SYSTEMS. *Wireless SoCs, Software, Cloud and AIoT Solutions* [online]. [B.r.]. Dostupné také z: https://www.espressif.com/.
5. ESPRESSIF SYSTEMS. *ESP32-C3 Technical Reference Manual* [online]. 2024. Ver. 1.1. Tech. zpr. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf.
6. ESPRESSIF SYSTEMS. *ESP32-S3 Technical Reference Manual* [online]. 2024. Ver. 1.5. Tech. zpr. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf.
7. ESPRESSIF SYSTEMS. *ESP32-C3 Series Datasheet* [online]. 2024. Ver. 1.7. Tech. zpr. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
8. ESPRESSIF SYSTEMS. *ESP32-S3 Series Datasheet* [online]. 2024. Ver. 1.8. Tech. zpr. Dostupné také z: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf.
9. ESPRESSIF SYSTEMS. *ESP-IDF Programming Guide* [online]. [B.r.]. Ver. 4.4.7. Dostupné také z: https://docs.espressif.com/projects/esp-idf/en/v4.4.7/esp32/index.html.
10. LINARO LIMITED. *MCUboot* [online]. 2023. Dostupné také z: https://docs.mcuboot.com/.
11. NORDIC SEMICONDUCTOR. *nRF Connect SDK Documentation* [online]. 2023. Ver. v2.5.1. Dostupné také z: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.5.1/nrf/index.html.
12. COLEMAN, Chris; MEMFAULT, INC. *MCUboot Walkthrough and Porting Guide* [online]. 2020. Dostupné také z: https://interrupt.memfault.com/blog/mcuboot-overview.
13. THE LINUX FOUNDATION. *Zephyr Project Documentation* [online]. 2023. Ver. 3.5.0. Dostupné také z: https://docs.zephyrproject.org/3.5.0/.

14. THE KERNEL DEVELOPMENT COMMUNITY. *Linux and the Devicetree* [online]. [B.r.]. Dostupné také z: https://www.kernel.org/doc/html/next/devicetree/usage-model.html.
15. THE KERNEL DEVELOPMENT COMMUNITY. *Kconfig Language* [online]. [B.r.]. Dostupné také z: https://www.kernel.org/doc/html/next/kbuild/kconfig-language.html.
16. LINARO LIMITED. *The Devicetree Specificationl* [online]. 2023. Ver. 0.4. Tech. zpr. Dostupné také z: https://www.devicetree.org/specifications/.
17. THE APACHE SOFTWARE FOUNDATION. *MCU Manager CLI* [online]. 2022. Dostupné také z: https://github.com/apache/mynewt-mcumgr-cli.
18. NORDIC SEMICONDUCTOR. *nRF Connect Device Manager* [online]. [B.r.]. Dostupné také z: https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-Device-Manager.
19. ARM HOLDINGS PLC. *ARM Software Development Toolkit Reference Guide* [https://developer.arm.com/]. [B.r.].

Obsah přílohy

- `readme.txt` ..... stručný popis obsahu média
- `zephyrproject` ..... zdrojové kódy implementace
- `thesis_src` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF