VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PROGRAMOVÁ VRSTVA PRO ŘÍZENÍ SPOTŘEBY VESTAVNÝCH SYSTÉMŮ

SOFTWARE LAYER FOR POWER MANAGEMENT OF EMBEDDED SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MÁRIO HARVAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

163000

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Harvan Mário, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vestavěné systémy
Název:	**Programová vrstva pro řízení spotřeby vestavných systémů**
Kategorie:	Vestavěné systémy
Akademický rok:	2024/25

Zadání:

1. Vytvořte přehled v oblasti řízení spotřeby elektrické energie ve vestavných systémech; zaměřte se zejména na systémy na bázi mikrokontrolérů (MCU).
2. Po dohodě s vedoucím zvolte sadu vestavných platforem na bázi MCU, se kterými budete dále pracovat. Pro každou z platforem detailně zdokumentujte dostupnou technickou (např. moduly a registry na čipu) a programovou (např. knihovny funkcí) podporu pro řízení spotřeby.
3. Shrňte požadavky kladené na (abstrakční) programovou vrstvu schopnou řídit spotřebu odlišných typů vestavných platforem na bázi MCU. Po dohodě s vedoucím využijte/upravte existující, popř. navrhněte vlastní, vrstvu coby základ řešení této práce.
4. Vrstvu z bodu 3 implementujte a pečlivě ji zdokumentujte (přehledově alespoň v Markdown formátu, detailněji pak pomocí vhodného (polo)automatizovaného generátoru dokumentace, např. Doxygen).
5. Využitelnost implementované vrstvy demonstrujte pomocí sady několika typově odlišných příkladů řízení spotřeby vestavných platforem na bázi MCU.
6. Zhodnoťte vlastnosti implementovaného řešení; identifikujte silné a slabé stránky řešení, navrhněte možné směry jeho vylepšení a rozveďte ty, které považujete za nejvíce perspektivní.

Literatura:

- AN13956: Power Manager Framework Usage for MCU Class of Devices. Application note, NXP. Available at https://www.nxp.com/docs/en/application-note/AN13956.pdf

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	31.10.2024

Abstrakt

Cieľom tejto práce je analyzovať problematiku riadenia spotreby moderných mikrokontrolérov a navrhnúť knižnicu na ich efektívne riadenie. Práca sa zameriava na definovanie požiadaviek, návrh a implementáciu knižnice riadenia spotreby, ktorá podporuje rôzne mikrokontroléry od popredných výrobcov. Funkčnosť knižnice bude overená pomocou testovacích sád, ktoré budú simulovať reálne aplikácie. Tieto testovacie sady budú zverejnené, čo umožní ostatným vývojárom porovnať výsledky s existujúcimi knižnicami a rôznymi typmi mikrokontrolérov.

Abstract

This thesis aims to analyze the power management issue in modern microcontrollers and to design a library for their efficient control. The work focuses on defining requirements, designing, and implementing a low-power management library that supports various microcontrollers from leading manufacturers. The library's functionality will be validated using test suites that simulate real-world applications. These test suites will be made publicly available, allowing other developers to compare the results with existing libraries and different types of microcontrollers.

Klíčová slova

vstavané systémy, mikrokontrolér, riadanie spotreby energie, režim spánku, knižnica na riadenie spotreby, meranie spotreby energie, STM32, NXP, ESP32, MCU

Keywords

embedded systems, low power, sleep modes, low power microcontrollers, power management, library, STM32, NXP, ESP32, MCU

Citace

HARVAN, Mário. *Programová vrstva pro řízení spotřeby vestavných systémů*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Programová vrstva pro řízení spotřeby vestavných systémů

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracovával samostatne pod vedením pána Ing. Josefa Strnadela, PhD. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Mário Harvan
16. května 2025

Poděkování

Týmto by som rád poďakoval vedúcemu mojej diplomovej práce Ing. Josefovi Strnadelovi, PhD., za odborné vedenie, čas venovaný konzultáciám a cenné informácie, ktoré mi pomohli pri realizácii tejto práce. Rovnako vyjadrujem vďaku aj svojim rodičom za ich neustálu podporu počas celého štúdia.

Obsah

Seznam obrázků

Zoznam skratiek

ADC Analog to Digital Converter

API Application Programming Interface

Armv7E ARM Cortex-M Architecture v7 Extension

AWIC Active Wake-up Interrupt Controller

BOR Brown-Out Reset

CAN Controller Area Network

CORDIC Coordinate Rotation Digital Computer

DAC Digital to Analog Converter

DMA Direct Memory Access

DSP Digital Signal Processor

EEMBC Embedded Microprocessor Benchmark Consortium

ESP32 Embedded System Platform 32-bit

FLASH Flash Memory

GPIO General Purpose Input/Output

HAL Hardware Abstraction Layer

HSI16 High-Speed Internal 16 MHz Oscillator

I2C Inter-Integrated Circuit

iRTC integrated Real-Time Clock

LLS Low Leakage Stop

LLS2 Low Leakage Stop 2

LLS3 Low Leakage Stop 3

LLWU Low-Leakage Wake-Up

LPM Library for Power Management

LPTIMER Low Power Timer

LPTMR Low Power Timer

LPUART Low Power UART

MCG Multipurpose Clock Generator

MCU Mikrokontrolér

MMU Memory Management Unit

NVIC Nested Vectored Interrupt Controller

OPAMP Operational Amplifier

PCB Printed Circuit Board

PLL Phase-Locked Loop

PMC Power Management Controller

POR Power On Reset

PSRAM Pseudo-Static Random Access Memory

PVD Power Voltage Detector

PWM Pulse Width Modulation

RAM Random Access Memory

RTC Real-Time Clock

RTOS Real-Time Operating System

SCR Status Control Register

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

SRPG State Retention Power Gating

SysTick System Timer

UART Universal Asynchronous Receiver/Transmitter

USB Universal Serial Bus

VBAT Battery Supply Voltage

VDD Voltage Drain Drain

VDDA Analog Power Supply Voltage

VLLS Very Low Leakage Stop

VLLS0 Very Low Leakage Stop 0

VLLS1 Very Low Leakage Stop 1

VLLS2 Very Low Leakage Stop 2

VLLS3 Very Low Leakage Stop 3

VLPR Very Low Power Run

VLPS Very Low Power Stop

VLPW Very Low Power Wait

WFE Wait For Event

WFI Wait For Interrupt

WIC Wake Up Interrupt Controller

Kapitola 1

Úvod

Vstavané systémy sú väčšinou nasadzované do prostredia, v ktorom je požadované dosiahnutie čo najnižšej spotreby. S nástupom a postupnou integráciou internetu vecí sa požiadavky rapídne zvyšovali. Od väčšiny zariadení je požadovaná dlhodobá funkčnosť s použitím typicky malého akumulátorového zdroja. V ideálnom prípade by takto nasadené zariadenia mali fungovať z jednej batérie po celú dobu svojej životnosti. Zlepšenia v účinnosti mikrokontrolérov a zefektívnenie výroby umožnili vyrobiť vstavané systémy, ktoré tieto požiadavky spĺňajú aj v komerčnej sfére. Väčšina výrobcov mikrokontrolérov videla v tomto novovzniknutom trhu potenciál na zvýšenie svojho podielu a začala navrhovať mikrokontroléry, ktoré podporujú komplexnejšie metódy na zníženie spotreby vstavaných systémov. Lídri v oblasti mikrokontrolérov taktiež začali navrhovať celé série mikrokontrolérov, ktorých hlavné zameranie je dosiahnuť čo najnižšiu možnú spotrebu v reálnych podmienkach.

Na riadenie spotreby mikrokontrolérov neexistuje žiadne štandardizované rozhranie, čo viedlo výrobcov k vývoju proprietárnych systémov. Moderné mikrokontroléry väčšinou využívajú komplexný systém riadenia spotreby, ktorý im umožňuje splniť požiadavky na spotrebu v širokom spektre rôznych prostredí. Nevýhodou komplexných systémov riadenia spotreby je, že sú v mnohých prípadoch príliš zložité pre vývojára, ktorý ich potrebuje využívať, čo predlžuje čas vývoja a zvyšuje jeho cenu. V prípade, že jeden vstavaný systém používa mikrokontroléry od rôznych výrobcov, je situácia ešte horšia. Každý výrobca má systém riadenia spotreby implementovaný rozdielnymi spôsobmi, čo znamená, že sa vývojár musí orientovať v každom z týchto systémov. Často sa stáva, že aj dve rozdielne rady mikrokontrolérov od toho istého výrobcu používajú odlišné systémy riadenia spotreby a aj rozhrania, cez ktoré sa ovládajú. Niektorí výrobcovia poskytujú knižnice, ktoré pomáhajú vývojárom pracovať so systémom riadenia spotreby. Na trhu je bohužiaľ veľmi málo knižníc, ktoré by podporovali mikrokontroléry od rôznych výrobcov.

Cieľom tejto práce je definovať unifikované rozhranie na prácu so systémami riadenia spotreby rôznych mikrokontrolérov a vyvinúť knižnicu, ktorá nebude závislá od konkrétnej platformy a bude jednoducho rozšíriteľná na rôzne druhy mikrokontrolérov. Teoretická časť práce sa zameriava na analýzu systémov riadenia spotreby populárnych mikrokontrolérov. Sú v nej objasnené základné princípy, ako dosiahnuť čo najnižšiu spotrebu pri zachovaní požadovanej funkcie vstavaného systému. Súčasťou práce je analýza dostupných knižníc, ktoré ponúkajú potrebnú funkcionalitu. Na základe dostupných riešení sú definované požiadavky na knižnicu riadenia spotreby, ktorá bude jednoducho rozšíriteľná a nezávislá od platformy konkrétneho výrobcu.

Kapitola 2

Problematika riadenia spotreby mikrokontrolérov

Cieľom tejto kapitoly je objasniť možnosti šetrenia spotreby energie aplikované v moderných mikrokontroléroch, analyzovať dostupné knižnice, ktoré umožňujú aplikačnej vrstve prácu s režimami nízkej spotreby a vytvoriť prehľad o rozhraniach, ktoré tieto knižnice ponúkajú. Moderné mikrokontroléry ponúkajú veľa mechanizmov na zníženie spotreby počas prevádzky mikrokontroléra, a taktiež množstvo režimov spánku. V tejto kapitole objasním všetky techniky zníženia spotreby mikrokontrolérov od vybraných populárnych výrobcov a analyzujem, ako dosahujú zníženie spotreby z hardvérového hľadiska.

2.1 Manažment spotreby mikrokontrolérov

V prostredí kde sa využívajú mikrokontroléry, je často kladený dôraz na nízku spotrebu energie. Niektoré aplikácie vyžadujú napájanie z akumulátorov alebo solárnych panelov a dlhú výdrž na jedno nabitie. Mikrokontrolér má od výrobcu udávanú spotrebu v aktívnom stave. Tú je možné znížiť vypnutím periférií, ktoré sa nepoužívajú, alebo znížením pracovnej frekvencie. Takéto zníženie spotreby nemusí byť dostatočné pre všetky aplikácie, preto mikrokontroléry podporujú rôzne režimy spánku, ktoré je možné riadiť z aplikačnej vrstvy. Ak je mikrokontrolér v režime spánku, tak je väčšinou jadro procesora úplne pozastavené, vďaka čomu má minimálnu spotrebu. Aktívne ostávajú iba periférie, ktoré sú nevyhnutne potrebné na činnosť systému, alebo sú použité na spracovanie externého signálu určeného na prebudenie mikrokontroléra. Väčšina mikrokontrolérov podporuje viacero úrovní spánku. Líšia sa hlavne tým, ktoré časti mikrokontroléra zostanú napájané. Pokiaľ sa jadru, alebo niektorej z periférií odpojí napájanie, tak sa stratí obsah všetkých registrov. Kvôli tomu sa pred prechodom do daného režimu spánku musí obsah registrov uložiť do nevolatilnej pamäte, alebo pamäte, ktorej napájanie ostane aktívne. Kvôli tomu môže trvať prechod medzi režimami spánku rôznu dobu a musí byť zohľadnený už pri výbere mikrokontroléra tak, aby vyhovoval požiadavkám danej aplikácie. Prechody do režimov spánku, v ktorých sa stratí obsah registrov, zvyšujú komplexnosť programu a predlžujú čas vývoja. Kvôli tomu je dôležité pred začiatkom vývoja analyzovať, či je dané zníženie spotreby naozaj nevyhnutné [5].

Na trhu dnes existujú špeciálne mikrokontroléry s nízkou spotrebou (napríklad séria STM32L4), ktoré ponúkajú viacero režimov spánku, umožňujúc programátorovi znižovať spotrebu v jemnejších krokoch. Tieto procesory podporujú aj stavy tvrdého spánku, v kto-

rých je typicky napájaná len malá časť pamäte Static Random Access Memory (SRAM) a zvyšok mikrokontroléra je úplne odpojený od napájania. Z týchto stavov sa dokážu zobudiť typicky len externým signálom na General Purpose Input/Output (GPIO) vstupe alebo pomocou Real-Time Clock (RTC) alarmu. Vďaka tomu môžu takéto mikrokontroléry dosahovať veľmi nízku spotrebu a z malého akumulátoru dokážu byť napájané aj niekoľko rokov [5].

2.1.1 Režimy spánku jadra ARM Cortex M4

Jadro ARM Cortex M4 je 32-bitové jadro používajúce architektúru ARM Cortex-M Architecture v7 Extension (Armv7E). Vďaka jeho nízkej cene a vysokej účinnosti je použité vo veľkej časti 32-bitových mikrokontrolérov. V experimentálnej časti tejto práce budeme testovať viacero mikrokontrolérov používajúcich toto jadro.

Toto jadro podporuje dva stavy nízkej spotreby. Jedná sa o stav spánku a stav hlbokého spánku. Jadro obsahuje Status Control Register (SCR), v ktorom je možné pomocou nastavenia jedného bitu zvoliť požadovaný stav, do ktorého má procesor prejsť. Jadro samotné nerozlišuje funkcionalitu medzi týmito dvomi stavmi. Je na výrobcovi mikrokontroléra, ako naimplementuje šetrenie energie v týchto stavoch. Obvyklý prístup je, že v režime spánku je pozastavený hlavný hodinový signál a príslušné obvody. V režime hlbokého spánku sa typicky, okrem hodinového signálu, odpojí aj napájanie procesora. Po nastavení príslušného bitu v registri SCR prejdeme do režimov spánku pomocou jednej z inštrukcií Wait For Event (WFE) — čakaj na udalosť, Wait For Interrupt (WFI) — čakaj na prerušenie. Inštrukcia WFE povolí procesoru prechod do režimu spánku, z ktorého sa zobudí, pokiaľ bude v event registry zapísaná nespracovaná udalosť. Inštrukcia WFI funguje podobne, s jediným rozdielom, že sa procesor zobudí vždy, keď nastane niektoré z povolených prerušení [16].

Kontext procesora v režimoch spánku

V režime normálneho spánku sa zastaví iba hodinový signál, napájanie jadra ostáva aktívne, takže sa zachová aj obsah všetkých registrov. Vďaka tomu je prechod medzi týmto a aktívnym režimom veľmi rýchly, ale úspora energie nie je najvyššia.

Pokiaľ sa v režime hlbokého spánku odpojí napájanie jadru procesora, obsah registrov sa vynuluje. Väčšina aplikácií požaduje, aby procesor po prebudení pokračoval z rovnakého stavu, v ktorom sa nachádzal pri prechode do hlbokého spánku. Tento problém je možné vyriešiť dvomi spôsobmi. Prvá možnosť je ukladať stav všetkých registrov pred prechodom do režimu hlbokého spánku do pamäte Random Access Memory (RAM). Po prebudení procesor načíta obsah registrov z RAM. Toto riešenie je z hardvérového hľadiska najjednoduchšie, ale predlžuje čas prechodu medzi týmito stavmi.

Druhá možnosť, ktorú podporuje jadro Cortex M4, je použiť špeciálne State Retention Power Gating (SRPG) klopné obvody. Tieto klopné obvody používajú dva zdroje napájania a po odpojení hlavného zdroja si zachovávajú svoj obsah. Hlavný zdroj, ktorý sa používa aj vo zvyšku jadra, je v režime hlbokého spánku odpojený. Druhý zdroj je záložný, väčšinou má veľmi nízky výkon a nízku spotrebu, ktorý napája len špeciálne klopné obvody a nikdy sa nevypína. Použitím klopných obvodov sa zrýchli prechod jadra medzi režimami spánku. Ich hlavnými nevýhodami sú väčšia komplexita, zaberajú väčšiu plochu na čipe a vyžadujú druhý zdroj napájania [1].

Prerušenia v režime hlbokého spánku

Jadro Cortex M4 obsahuje v základnej konfigurácii len jeden kontrolér prerušení Nested Vectored Interrupt Controller (NVIC). Pokiaľ sa v režime hlbokého spánku jadru odpojí napájanie, tento kontrolér prerušení prestane fungovať a procesor nebude možné prebudiť pomocou bežných prerušení. V tejto konfigurácii je možné procesor prebudiť iba špeciálnym Wake-Up signálom, ktorý môže byť generovaný externou perifériou. Môže sa použiť napríklad externé RTC s funkciou alarmu. Toto riešenie značne limituje flexibilitu využitia režimu hlbokého spánku, keďže nie je možné procesor prebudiť žiadnou z periférií mikrokontroléra [4].

Cortex M4 je v návrhu možné nakonfigurovať tak, aby obsahoval špeciálny kontrolér prerušení Wake Up Interrupt Controller (WIC). Tento kontrolér obsahuje kópiu niektorých registrov z NVIC a umožňuje procesor zobudiť z režimu hlbokého spánku prerušením, aj keď má zvyšok jadra odpojené napájanie. WIC je jednoduchší, nepodporuje všetky nastavenia z NVIC, ale vie spracovať väčšinu prerušení. Pri prechode do hlbokého spánku procesor automaticky prekopíruje nastavenia z NVIC do WIC. WIC neobsahuje žiadne konfigurovateľné registre, jeho činnosť je plne automatická. Jadro Cortex M4 je možné nakonfigurovať tak, aby obsahovalo kontrolér prerušení WIC, takže sa môže výrobca mikrokontroléru rozhodnúť, či ho chce používať [1].

2.1.2 Mikrokontrolér STM32G4

Mikrokontrolér STM32G4 využíva jadro ARM Cortex M4. Obsahuje veľké množstvo periférií, presných a rýchlych Analog to Digital Converter (ADC) prevodníkov, modul Coordinate Rotation Digital Computer (CORDIC) na hardvérovú akceleráciu trigonometrických funkcií, vstavané RTC, ktoré je možné napájať z externej batérie a prepracované možnosti úspory energie. Aktuálne je tento mikrokontrolér populárny a aktívne nasadzovaný do rôznych aplikácií. Je to ideálny kandidát pre aplikácie, ktoré vyžadujú vysoký výpočtový výkon, veľa možností konfigurácie a nízku spotrebu v úsporných režimoch [12].

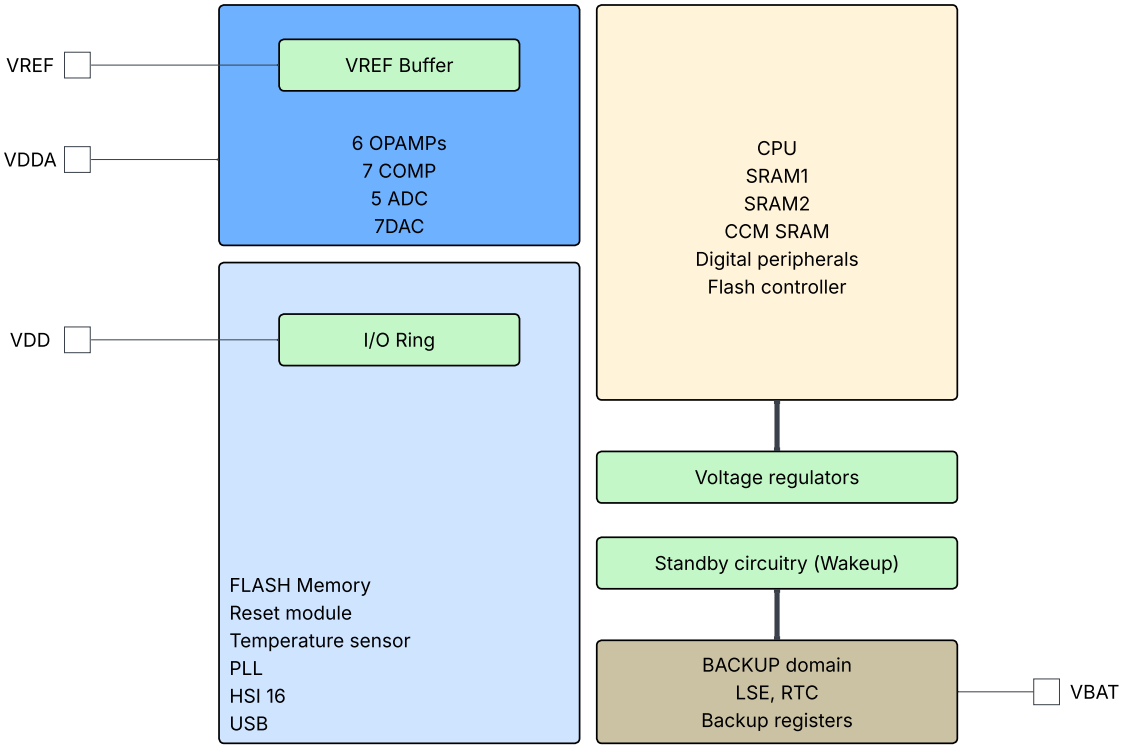
Napájacie obvody mikrokontroléru

Moderné mikrokontroléry obsahujú prepracovanú sústavu napájacích obvodov, čo je dôležité pre dosiahnutie nízkej spotreby energie. Rôzne zdroje napájania umožňujú programátorovi vypnúť nepotrebné časti mikrokontroléra v režimoch spánku, alebo aktivovať napäťový regulátor s nižšou spotrebou.

Ako hlavný zdroj sa používa Voltage Drain Drain (VDD), ktorý napája všetky periférie okrem RTC, Flash Memory (FLASH) pamäť a všetky generátory hodinového signálu. Napájanie VDD podporuje vstupné napätie 1.71 V až 3.6 V. To umožňuje návrhárovi použiť vhodné vstupné napätie podľa požiadaviek aplikácie tak, aby zariadenie mohlo dosiahnuť čo najvyššiu efektivitu.

Druhým nezávislým zdrojom je Analog Power Supply Voltage (VDDA), ktoré sa používa na napájanie ADC, Digital to Analog Converter (DAC) a Operational Amplifier (OPAMP) prevodníkov. Vďaka oddelenému zdroju môžu tieto periférie pracovať aj s väčším napätím, ako je napätie na vstupe VDD.

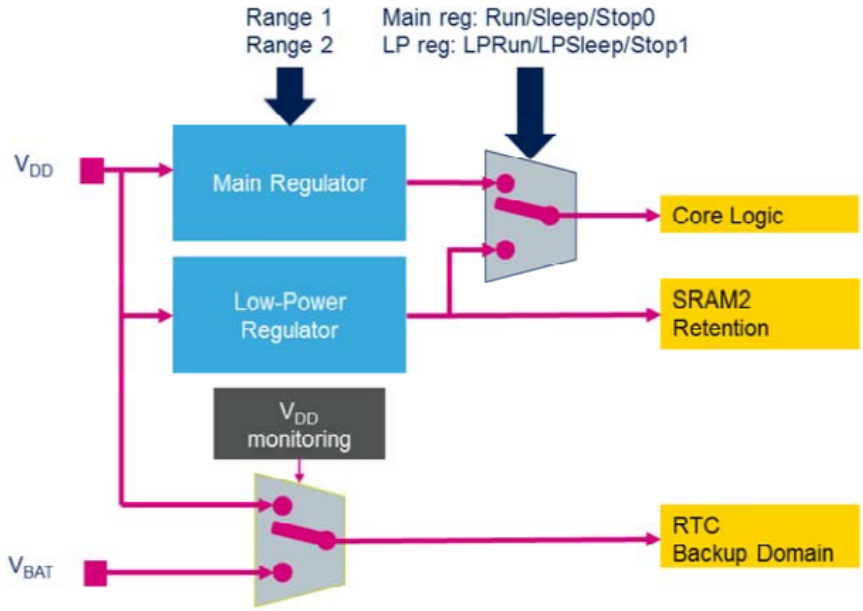
Poslednou napäťovou doménou je Backup, ktorá je napájaná zo zdroja Battery Supply Voltage (VBAT). Ten je určený na pripojenie externého akumulátora. Z neho je napájaná periféria RTC a backup register.



Obrázek 2.1: Bloková schéma napájacích obvodov mikrokontroléru STM32G4. Zdroj: STMicroelectronics (2020) [11]

Ďalšou dôležitou časťou je supervízor zdroja napájania, ktorý garantuje bezpečný stav a reštartovanie Mikrokontrolér (MCU) v prípade poklesu napätia na napájacom vstupe. Supervízor podporuje Power On Reset (POR), ktorý reštartuje MCU po spustení a zaistí, že sa spustil až keď sa napájacie napätie stabilizovalo. Taktiež podporuje Brown-Out Reset (BOR), ktorý MCU reštartuje, keď zachytí pokles napätia pod zvolenú konfigurovateľnú úroveň. Poslednou časťou je modul Power Voltage Detector (PVD), ktorý vie vygenerovať prerušenie, keď napätie poklesne pod užívateľom zvolenú úroveň. To môže byť užitočné napríklad ako upozornenie tesne pred vypnutím MCU, ktorý môže ešte vykonať potrebné úkony [12].

Napäťové regulátory sa používajú na zníženie a stabilizovanie napätia pre každú časť mikrokontroléra. Obecne platí, čím viac výkonu dokáže regulátor dodať, tým menej efektívny bude pri nízkom zaťažení. Kvôli tomu má STM32G4 hlavný regulátor, ktorý sa používa, keď má procesor nastavenú vysokú frekvenciu hodinového signálu. Regulátor taktiež podporuje dynamické škálovanie napätia. S rastúcou frekvenciou hodinového signálu zvyšuje napätie, čo umožňuje procesoru spoľahlivo pracovať na vyššej frekvencii za cenu vyššej spotreby. Druhý je regulátor s nízkym výkonom, ktorý sa používa na napájanie pamäte SRAM2 a taktiež môže napájať procesor v režime spánku, alebo ak má hodinový signál nižšiu frekvenciu [12].



Obrázek 2.2: Bloková schéma napäťových regulátorov STM32G4. Zdroj: STMicroelectronics (2020) [11]

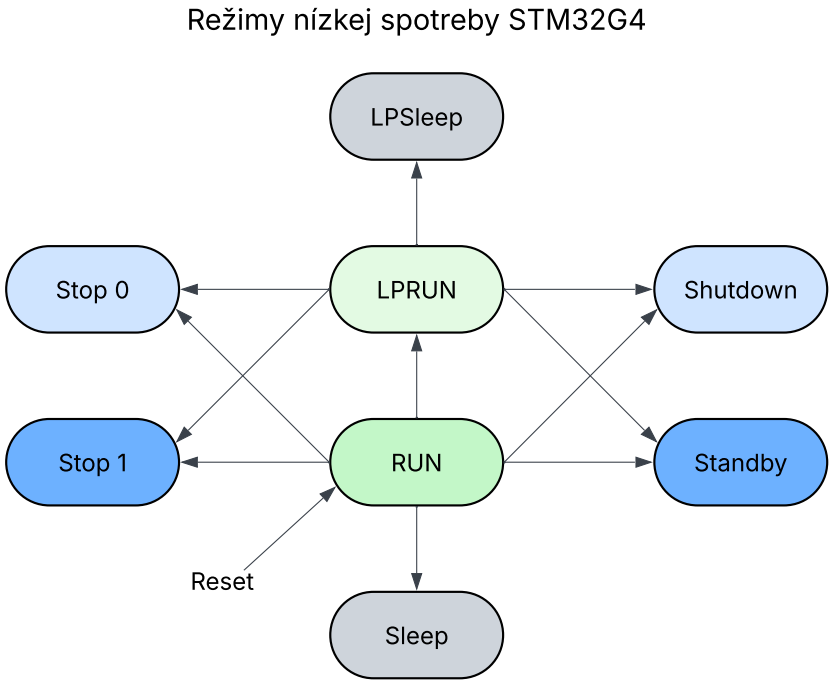
Režimy prevádzky mikrokontroléru

Mikrokontrolér podporuje viacero režimov prevádzky, v ktorých je program spustený z pamäte FLASH alebo SRAM, režimy spánku, režimy stop, standby a režim úplného vypnutia napájania pre mikrokontrolér. Jednotlivé režimy sa líšia v dosiahnutej spotrebe a frekvencii hodinového signálu. Najväčšie rozdiely medzi režimom spánku a standby sú doba, za ktorú sa dokáže mikrokontrolér z daného režimu prebudiť, a počet aktívnych periférií. Podľa zvoleného režimu spánku a počtu aktívnych periférií sa mení aj spotreba mikrokontroléra.

V režimoch aktívnej prevádzky sa vykonávaný kód nachádza v FLASH pamäti. Mikrokontrolér podporuje tri aktívne režimy (RUN 1 boost, RUN 1, RUN 2), v najrýchlejšom režime môže dosiahnuť frekvenciu až 170 MHz, v režime RUN1 frekvenciu 150 MHz a v režime RUN2 26 MHz. Rozdiel v spotrebe medzi jednotlivými režimami a frekvenciami môže byť pomerne veľký, preto môže byť výhodné optimalizovať program tak, aby bolo možné čo najviac znížiť frekvenciu hodinového signálu. Mikrokontrolér podporuje ešte špeciálny režim LPRUN, v ktorom môže byť maximálny takt 2 MHz. V tomto režime má mikrokontrolér vypnutý hlavný napájací regulátor a aktívny je len regulátor s nízkym výkonom. Program musí byť uložený v pamäti SRAM1, ktorá má obmedzenú veľkosť. Výhodou je značné zníženie spotreby oproti ostatným režimom. V každom z režimov aktívnej prevádzky je možné odpojiť pamäť FLASH od zdroja napájania a program uložiť do pamäte SRAM1, čo pomôže znížiť spotrebu. Taktiež je možné vypínať hodinový signál pre každú z periférií. Programátor môže zapnúť len tie periférie, ktoré nutne potrebuje, čo pomôže znížiť spotrebu energie. Mikrokontrolér sa môže medzi jednotlivými režimami dynamicky prepínať počas behu programu. Ak aplikácia potrebuje vysoký výpočtový výkon, môže sa na chvíľu prepnúť do režimu s vyššou frekvenciou. Po dokončení náročného výpočtu môže znovu prejsť do úsporného režimu [12].

Režimy spánku mikrokontroléra

V režime spánku sa vypne procesor Cortex M4, hodinový signál vstupujúci do jednotlivých periférií môže ostať pripojený alebo odpojený podľa nastavenej konfigurácie v softvéri. Mikrokontrolér podporuje režim Sleep a Low power sleep. Z týchto režimov spánku sa mikrokontrolér dokáže prebudiť najrýchlejšie. V režime sleep podporuje frekvenciu hodinového signálu až 170 MHz pre periférie. Hodinový signál pre pamäť SRAM ostáva aktívny, takže sa po prebudení zachová celý obsah pamäte. Mikrokontrolér je možné prebudiť vyvolaním prerušenia zo všetkých periférií, ktoré ostali v tomto režime zapnuté [4].



Obrázek 2.3: Možnosti prechodov medzi režimami prevádzky STM32G4. Zdroj: STMicroelectronics (2020) [11]

V režime Low power sleep je procesor taktiež vypnutý. Zvyšok systému je napájaný z regulátora s nízkou spotrebou. Maximálna frekvencia hodinového signálu v tomto režime je 2 MHz. Napájanie pamäte FLASH je v tomto režime konfigurovateľné. Môže byť odpojené pre nižšiu spotrebu alebo pripojené pre rýchlejšie prebudenie systému. Všetky periférie okrem Universal Serial Bus (USB) môžu ostať aktívne.

Režimy spánku sú preto ideálne na jednoduché ušetrenie energie, keďže všetky potrebné periférie môžu ostať aktívne, nie je potrebné ukladať dáta z pamäte SRAM a mikrokontrolér sa dokáže rýchlo prebudiť. Pre programátora je prechod do tohto režimu najjednoduchší na implementáciu [4].

Prechod do režimu spánku je pomocou knižnice Hardware Abstraction Layer (HAL) jednoduchý. V prvom rade je potrebné vypnúť System Timer (SysTick) prerušenie, ktoré je generované každú milisekundu na udržiavanie interného počítadla. Ak by toto prerušenie nebolo vypnuté, tak by sa mikrokontrolér zobudil každú milisekundu. Následne stačí zavolať funkciu na prechod do režimu spánku. V parametroch môžeme špecifikovať, aký zdroj napájania sa má v spánku používať. Mikrokontrolér sa prebudí po prijatí jedného z nakonfigurovaných prerušení. Pokiaľ sú povolené prerušenia, z ktorých sa mikrokontrolér

nemá zobudiť, je ich potrebné vypnúť ešte pred prechodom do režimu spánku pomocou masky [12].

```
// Vypnutie systick prerušenia
HAL_SuspendTick();
// Prechod do režimu spánku so zapnutým hlavným regulátorom napájania
HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
```

Ak má mikrokontrolér po prebudení pokračovať vo vykonávaní programu, je potrebné obnoviť systick prerušenie.

```
HAL_ResumeTick(); // Opätovné povolenie prerušenia
```

Ďalšou zaujímavou možnosťou, ktorú mikrokontrolér podporuje, je opätovný prechod do režimu spánku po spracovaní prerušenia. Túto funkciu je možné využiť, ak má mikrokontrolér spracovať nejaký vstup z periférie a znovu prejsť do režimu spánku. Pre obnovenie normálnej prevádzky je potrebné túto funkciu deaktivovať po prebudení mikrokontroléra [12].

```
HAL_PWR_EnableSleepOnExit(); // Zapnutie funkcie prechodu spánku po spracovaní
↪  prerušenia
HAL_PWR_DisableSleepOnExit(); // Vypnutie automatického prechodu do režimu
↪  spánku
```

Režimy zastavenia mikrokontroléra

STM32G4 podporuje režim STOP 0 a STOP 1. Sú to režimy, ktoré dosiahnu najnižšiu spotrebu pri zachovaní obsahu v pamäti SRAM a všetkých registrov periférií. Všetky vysokofrekvenčné hodinové signály sú v týchto režimoch deaktivované. Niektoré z periférií môžu ostať aktívne a umožňujú prebudiť systém vygenerovaním prerušenia.

V režime stop 0 ostáva aktívny hlavný regulátor napätia, ktorý napája potrebné pamäte, registre a aktívne periférie. Periférie Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver/Transmitter (UART), Low Power UART (LPUART), Low Power Timer (LPTIMER), RTC môžu v tomto režime ostať aktívne. Pre I2C a UART je možné nakonfigurovať očakávaný obsah prijatej správy, po ktorej majú vygenerovať prerušenie na prebudenie zvyšku systému. Kvôli podpore tejto funkcie dokážu periférie samy aktivovať hodinový signál High-Speed Internal 16 MHz Oscillator (HSI16), vďaka ktorému dokážu správu dekódovať a následne ho znovu vypnúť a čakať na ďalšiu správu.

Hlavný rozdiel medzi režimami STOP 0 a 1 je použitý regulátor napätia. V režime STOP 1 je vypnutý hlavný regulátor napätia a používa sa regulátor s nízkou spotrebou energie. Vďaka tomu dosahuje nižšiu spotrebu, ako režim STOP 0. Hlavnou nevýhodou je dlhšia doba, za ktorú sa systém dokáže prebudiť. V režime STOP 0 je doba prebudenia približne 3 µs, v režime STOP 1 približne 9 µs.

Prechod do režimu zastavenia je podobný, ako do režimu spánku. Najskôr je potrebné vypnúť SysTick prerušenie, a následne zavolať funkciu z knižnice HAL na prechod do režimu

zastavenia. Pomocou parametrov funkcie je možné zvoliť, aký napäťový regulátor sa má použiť [12].

```
// Vypnutie systick prerušenia
HAL_SuspendTick();
HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI);
```

Po prebudení z režimu zastavnia je potrebné znovu nakonfigurovať hodinový signál pre všetky periférie a procesor. Pokiaľ sa na konfiguráciu hodinového signálu využíva knižnica HAL, opätovná konfigurácia je výrazne jednoduchšia.

```
SystemClock_Config(); // Konfigurácia hodinového signálu v knižnici HAL
HAL_ResumeTick();     // Aktivácia systick prerušenia
```

V režime zastavenia je možné použiť funkciu na opätovný prechod do režimu zastavenia po spracovaní prerušenia, rovnako ako v režime spánku. [12]

Pohotovostný režim mikrokontroléra

Pohotovostný režim dosahuje najnižšiu spotrebu pri zachovaní 16 KB obsahu pamäte SRAM2. V tomto režime dokáže mikrokontrolér automaticky prepnúť zdroj napájania z hlavného na záložný batériový zdroj. To umožňuje návrhárovi napájacej sústavy úplne vypnúť všetky zdroje napájania a znížiť spotrebu obvodov okolo mikrokontroléra. V štandardnej konfigurácii sa vypnú všetky interné zdroje napájania MCU a stratí sa obsah všetkých registrov a aj pamäte SRAM. Konfiguráciou je možné nechať aktívny zdroj s nízkou spotrebou, ktorý napája pamäť SRAM2. V oboch konfiguráciách procesor zachová 128 bajtov veľký záložný register, do ktorého je možné uložiť dôležité informácie. Pred prechodom do pohotovostného režimu je možné pre každý GPIO vstup nastaviť napájací alebo zemniaci rezistor, ktorý ostane pripojený po celú dobu spánku.

Mikrokontrolér je možné prebudiť pomocou privedenia signálu na jeden z piatich vstupov podporujúcich zobudenie. Periférium RTC je možné napájať zo záložného zdroja napájania. Táto periféria taktiež dokáže procesor prebudiť pomocou nakonfigurovaného alarmu.

Pri prechode do pohotovostného režimu sa musia najskôr vypnúť všetky príznaky, ktoré by mikrokontrolér mohli zobudiť. Následne je potrebné nakonfigurovať niektorý zo vstupov, ktorý podporuje prebudenie. Ak nebude nakonfigurovaný žiadny zo vstupov, tak sa mikrokontrolér nikdy z pohotovostného režimu neprebudí [12].

```
// Vymazanie príznakov na prebudenie mikrokontroléra
__HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
// Vymazanie príznakov na prebudenie pomocou periférie RTC
__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG(&hrtc, RTC_FLAG_WUTF);
// Nastavenie prebudenia pomocou vstupného signálu na pine 1
HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1);

// Prechod do pohotovostného režimu
HAL_PWR_EnterSTANDBYMode();
```

Obnovenie z pohotovostného režimu je zložitejšie, keďže sa neuložil obsah pamäte SRAM. Procesor začína vykonávať program od začiatku tak, ako pri prvotnom zapnutí. Jediný spôsob, ako rozlíšiť prebudenie z pohotovostného režimu od normálneho zapnutia, je jeden z príznakov v nevolatilnom registri. Pokiaľ je nastavený, musí sa vynulovať, a taktiež je nutné deaktivovať nakonfigurovaný vstup na prebudenie mikrokontroléra.

```
if (__HAL_PWR_GET_FLAG(PWR_FLAG_SB) != RESET)
{
    // Vymazanie príznaku prebudenia
    __HAL_PWR_CLEAR_FLAG(PWR_FLAG_SB);
    // Deaktivácia prebudenia na vstupe 1
    HAL_PWR_DisableWakeUpPin(PWR_WAKEUP_PIN1);
}
```

Režim vypnutia mikrokontroléra

V režime vypnutia mikrokontrolér dosahuje najnižšiu spotrebu zo všetkých režimov. Všetky periférie aj procesor sú vypnuté a žiadna z pamätí si nezachová svoj obsah. Mikrokontrolér je možné prebudiť len pomocou piatich vstupov, ktoré prebudenie podporujú. Najväčší rozdiel oproti pohotovostnému režimu je, že v režime vypnutia nie sú aktívne ochrany na pokles vstupného napätia. To znamená, že v prípade poklesu vstupného napätia pod 1.6 V nie je garantované zachovanie správnej funkcionality mikrokontroléru. Tento režim je bezpečné používať len ak je vstavaný systém vybavený spoľahlivým záložným zdrojom. V tomto režime má mikrokontrolér odber prúdu len 15 nA [12].

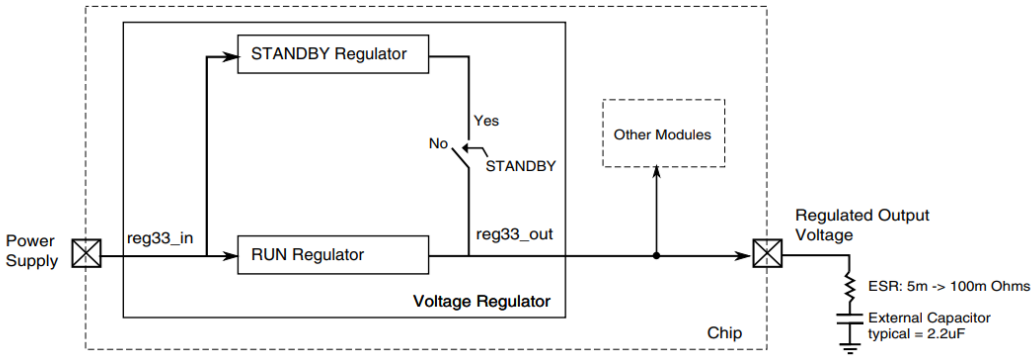
2.1.3 Mikrokontrolér NXP Kinetis K60

Mikrokontrolér NXP Kinetis K60 patrí do rodiny vysoko výkonných 32-bitových mikrokontrolérov založených na architektúre ARM Cortex M4. Tento mikrokontrolér je navrhnutý pre širokú škálu aplikácií, ktoré vyžadujú vysoký výpočtový výkon, efektívnu správu energie a pokročilé moduly periférií. Vďaka podpore hardvérovej jednotky na spracovanie signálov Digital Signal Processor (DSP) a modulu plávajúcej desatinnej čiarky je Kinetis K60 ideálny kandidát pre aplikácie v priemyselných systémoch, inteligentných zariadeniach a automobilovom priemysle. Mikrokontrolér ponúka bohatú sadu periférií, ako sú rozhrania Ethernet, USB, Controller Area Network (CAN) a ADC, ktoré umožňujú jeho integráciu do komplexných systémov. Okrem toho poskytuje pokročilé bezpečnostné funkcie, ako je kryptografia a ochrana proti neoprávnenému prístupu, čím zvyšuje úroveň spoľahlivosti a bezpečnosti aplikácií. Dalo by sa povedať, že tento mikrokontrolér je konkurenciou pre MCU STM32G4 firmy ST Electronics. Sú postavené na rovnakom jadre a podporujú podobné periférie. Preto som sa rozhodol implementovať jeho podporu do knižnice na riadenie spotreby energie. Bude zaujímavé porovnať, ktorý dosahuje nižšiu spotrebu, a určiť, ktorý má prepracovanejšiu podporu režimov pre zníženie spotreby [7].

Napájacie obvody mikrokontroléru

Mikrokontrolér Kinetis K60 má jednoduchšiu schému napájacích regulátorov, ako STM32G4. Obsahuje jeden hlavný regulátor napätia, ktorý podporuje vstupné napätie od 2.7 V do 5.5 V.

Tento regulátor sa používa vo všetkých stavoch normálnej operácie mikrokontroléra. Pokiaľ však prejde do jedného z režimov spánku, hlavný regulátor sa odpojí a využíva sa pohotovostný regulátor. Ten dodáva menší výkon, ale dosahuje väčšiu účinnosť. Oba regulátory je možné vyradiť a mikrokontrolér sa môže napájať priamo zo vstupného napätia [6].



Obrázek 2.4: Bloková schéma napájacích regulátorov Kinetis K60.
Zdroj: NXP Semiconductors (2015) [7]

Podporované je taktiež napájanie zo záložného zdroja VBAT, ktoré napája periférium RTC a integrated Real-Time Clock (iRTC). Obe periférie môžu zostať aktívne aj po úplnom odpojení hlavného napájania z mikrokontroléra a udržiavať tak napríklad aktuálny čas, alebo zobudiť mikrokontrolér po vypršaní nastaveného časovača.

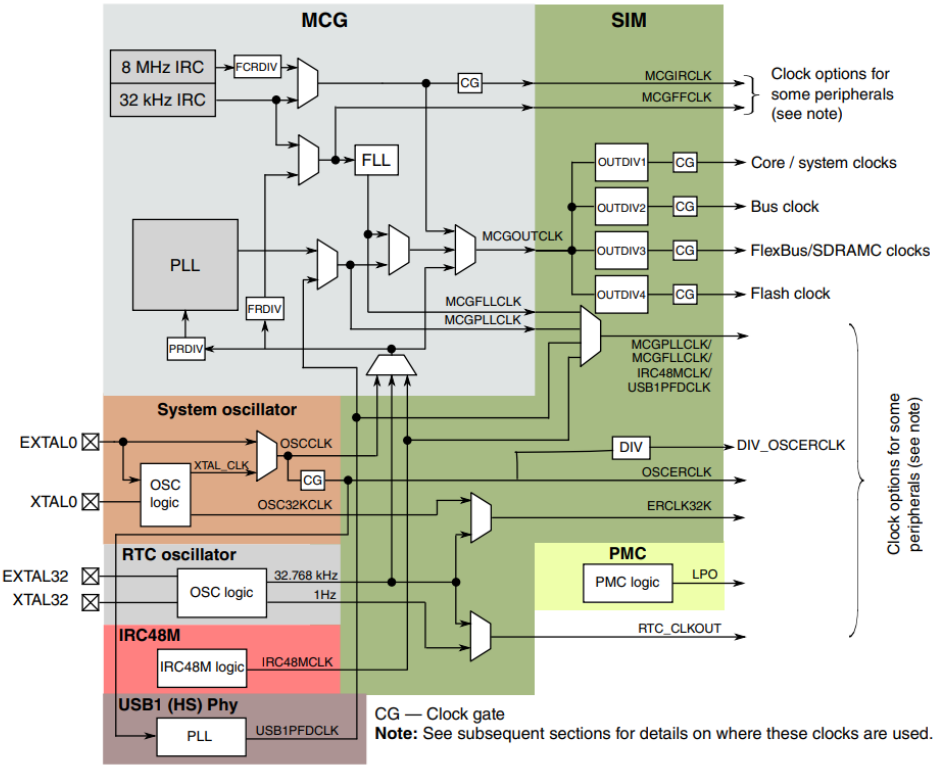
Mikrokontrolér obsahuje modul na detekciu nízkeho vstupného napätia. Tento modul je dôležitý v prípade, že vstupné napätie poklesne pod minimálnu hodnotu stanovenú výrobcom. V tomto prípade modul zaručí, že sa mikrokontrolér po obnovení napájania reštartuje a obnoví sa validný stav všetkých registrov a periférií. Modul má taktiež konfigurovateľnú hranicu napätia, kedy vygeneruje prerušenie. To sa môže použiť, keď mikrokontrolér musí vykonať dôležitú funkciu, napríklad uvedenie systému do bezpečného stavu, predtým, než sa vypne [6].

Generátory hodinového signálu

Prepracovaný obvod generátorov hodinového signálu je kľúčový na zníženie spotreby mikrokontroléra v režimoch spánku. Obecne platí, čím vyššiu frekvenciu generátor má, tým má väčšiu spotrebu. Moderné mikrokontroléry obsahujú viacero generátorov hodinového signálu s rôznymi frekvenciami a užívateľovi umožňujú prepínať medzi nimi podľa potreby aplikácie.

Kinetis K60 ponúka prepracovaný systém generátorov hodinového signálu. Hlavným modulom je Multipurpose Clock Generator (MCG), ktorý dokáže generovať signál s frekvenciou 1 MHz až 100 MHz. Počas normálnej prevádzky mikrokontroléra všetky periférie a aj jadro Cortex M4 využívajú hodinové signály z tohto modulu. Periféria RTC využíva RTC oscilátor. Dôležité je, že tento zdroj hodinového signálu je oddelený od ostatných, a teda umožňuje generovať hodinový signál na zachovanie aktuálneho času aj v prípade vypnutia všetkých ostatných oscilátorov.

Modul oscilátorov Power Management Controller (PMC) obsahuje oscilátor s nízkou spotrebou energie. Ten sa využíva ako zdroj hodinového signálu pre periférie, ktoré podporujú režim nízkej spotreby, v prípade, že je jadro mikrokontroléra v režime spánku.



Obrázek 2.5: Schéma generátorov hodinového signálu mikrokontroléru Kinetis K60. Zdroj: NXP Semiconductors (2015) [7]

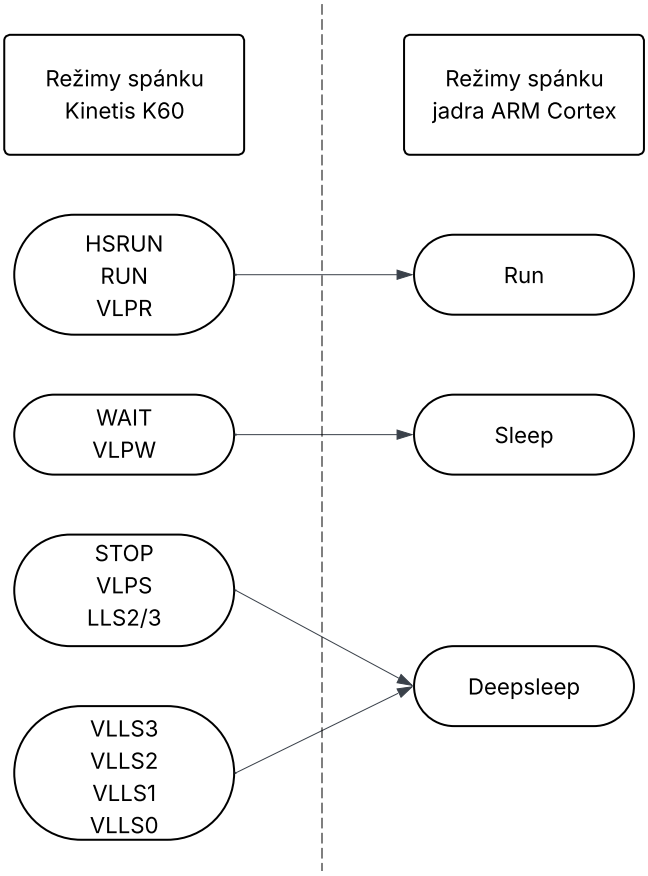
Modul SIM umožňuje odpájať a pripájať jednotlivé časti mikrokontroléra k zdrojom hodinového signálu. Zapisovaním do ovládacieho registra modulu môže užívateľ odpájať jednotlivé periférie od oscilátorov a tým znižovať spotrebu počas prevádzky a taktiež v režimoch spánku. Čím viac možností tento modul ponúka, tým je možné lepšie prispôsobiť spotrebu mikrokontroléra pre rôzne aplikácie vypínaním periférií, ktoré sa nepoužívajú [7].

Režimy aktívnej prevádzky mikrokontroléru

Mikrokontrolér podporuje tri režimy bežnej prevádzky: základný, vysoko-rýchlostný a režim nízkej spotreby Very Low Power Run (VLPR). V týchto režimoch je program spustený z pamäte FLASH, jadro mikrokontroléra je aktívne a spracováva inštrukcie. Všetky periférie, ku ktorým je pripojený hodinový signál, sú aktívne.

V základnom režime sa mikrokontrolér nachádza po spustení. Na napájanie sa využíva hlavný napäťový regulátor. Maximálna frekvencia procesora v tomto režime je 100 MHz.

V režime nízkej spotreby VLPR je frekvencia procesora a zberníc limitovaná na 4 MHz. Napäťový regulátor sa nastaví podľa aktuálnej spotreby mikrokontroléra. Tá závisí od počtu periférií, ktoré ostali aktívne. V prípade nízkej spotreby systém automaticky aktivuje regulátor s nízkou spotrebou. Zaujímavé je, že zapisovanie do pamäte FLASH nie je v tomto režime prevádzky podporované [6].



Obrázek 2.6: Režimy prevádzky mikrokontroléra Kinetis K60 a jadra ARM Cortex. Zdroj: NXP Semiconductors (2015) [7]

Režimy spánku

Kinetis K60 obsahuje jadro Cortex M4, ktoré používa modul NVIC na spracovanie prerušení. V režimoch spánku ostáva modul NVIC aktívny a dokáže spracovať všetky prerušenia rovnako, ako v režime aktívnej prevádzky. Konfigurácia jadra Cortex M4 umožňuje pridať jednotku Active Wake-up Interrupt Controller (AWIC), ktorá dokáže spracovávať asynchrónne prerušenia a nepotrebuje na funkciu hodinový signál. V mikrokontroléri Kinetis K60 sa používa na spracovanie vybraných prerušení v režimoch hlbokého spánku [7].

Kinetis K60 podporuje nasledujúce režimy spánku Wait a Very Low Power Wait (VLPW) režim. V režime Wait je jadro mikrokontroléra uspané a má odpojený hodinový signál. Modul prerušení NVIC ostáva aktívny, takže v tomto režime podporuje prebudenie všetkými bežnými prerušeniami. Hodinový signál pre periférie ostáva v rovnakej konfigurácii, ako v režime aktívnej činnosti mikrokontroléra. Užívateľ môže znížiť spotrebu vypnutím periférií, ktoré sa nevyužívajú na prebudenie mikrokontroléra pred prechodom do režimu spánku. Napájanie pre pamäť SRAM ostáva aktívne, takže sa obsah pamäte zachová. Rovnako sa zachová obsah všetkých registrov jadra a taktiež periférií. To umožňuje rýchle prebudenie po prijatí prerušenia.

VLPW režim je podobný, ako Wait režim. Jadro Cortex M4 je v režime spánku s odpojeným hodinovým signálom. Modul prerušení NVIC ostáva aktívny. Napäťový regulátor sa automaticky prepne do príslušného režimu podľa aktuálnej konfigurácie periférií. Najväčším

obmedzením tohto režimu je, že mikrokontrolér do neho môže vstúpiť iba z režimu VLPR, čo značne obmedzuje jeho využitie počas bežného použitia mikrokontroléra.

V prípade vygenerovaného prerušenia mikrokontrolér prejde do režimu aktívnej prevádzky. Nie je potrebné znovu konfigurovať žiadne periférie, keďže všetky boli v režime spánku aktívne [7].

Režimy zastavenia

Mikrokontrolér podporuje 6 režimov zastavenia: režim zastavenia, režim zastavenia s veľmi nízkou spotrebou Very Low Power Stop (VLPS), režimy zastavenia s nízkym únikom prúdu Low Leakage Stop (LLS), Low Leakage Stop 2 (LLS2), Low Leakage Stop 3 (LLS3) a režimy zastavenia s veľmi nízkym únikom prúdu Very Low Leakage Stop 3 (VLLS3), Very Low Leakage Stop 2 (VLLS2), Very Low Leakage Stop 1 (VLLS1), Very Low Leakage Stop 0 (VLLS0).

V režime zastavenia je jadro mikrokontroléra v režime hlbokého spánku s odpojením hodinového signálu. Modul prerušení NVIC je vypnutý a namiesto neho sa používa modul WIC, ktorý dokáže mikrokontrolér prebudiť po príchode jedného z podporovaných prerušení. Hodinový signál je pre väčšinu periférií pozastavený. Direct Memory Access (DMA) periféria môže byť aktívna aj v režimoch zastavenia a môže sa využívať na prenos dát z periférií do pamäti SRAM. Pamäť SRAM zostáva napájaná a zachováva si celý svoj obsah aj po prebudení. Modul detekcie nízkeho vstupného napätia je taktiež aktívny a umožňuje vygenerovať prerušenie v prípade poklesu napätia pod nakonfigurovanú hranicu.

Režim VLPS je rovnaký, ako režim normálneho zastavenia. Jediným rozdielom je konfigurácia napäťových regulátorov, ktorá by mala mať nižšiu spotrebu, a prebudenie z tohto režimu trvá dlhší čas.

Režimy zastavenia LLS, LLS2 a LLS3 sú si veľmi podobné. Vo všetkých režimoch je jadro v režime hlbokého spánku s odpojeným hodinovým signálom. Modul NVIC aj WIC sú pozastavené. Dostupný je iba špeciálny modul prerušenia Low-Leakage Wake-Up (LLWU), ktorý podporuje prebudenie iba z vybraných externých vstupných signálov a z RTC periférie. Všetky periférie majú odpojený hodinový signál, obsah ich registrov napriek tomu ostane zachovaný. V režime LLS3 uložený kontext celej pamäte SRAM, v režimoch LLS2 a LLS3 sa udrží jej časť. Po prebudení a spracovaní prerušenia z modulu LLWU bude MCU pokračovať vo vykonávaní nasledujúcej inštrukcie, ktorú spracoval pred prechodom do režimu spánku. Výhodou tohto mikrokontroléra oproti konkurencii je, že konfigurácia periférií ostane po prebudení zachovaná, a vývojár ju nemusí opakovane spúšťať.

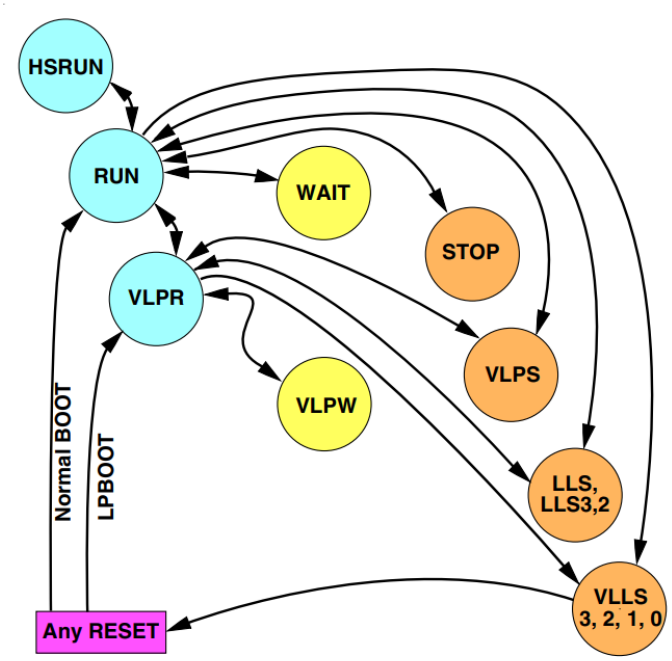
Režim zastavenia s veľmi nízkym únikom prúdu VLLS3 je podobný režimu LLS. Celý obsah pamäte SRAM je zachovaný. Hlavným rozdielom je, že po prebudení mikrokontrolér nepokračuje v rozpracovanom toku programu, reštartuje sa a program začína vykonávať odznova. To znamená, že je nutné opätovne inicializovať všetky periférie a celú inicializáciu aplikačnej vrstvy.

Režim VLLS2 je veľmi podobný režimu VLLS3. Jediným rozdielom je, že sa zachová obsah iba 4 kB časti pamäte SRAM. Režim VLLS1 nezachováva žiadnu časť pamäte SRAM. Modul na kontrolu vstupného napätia ostáva v týchto režimoch aktívny. Taktiež je možné mikrokontrolér prebudiť jedným z externých vstupných signálov a perifériou RTC.

Hlavný rozdiel režimu VLLS0 je, že modul na kontrolu vstupného napätia môže byť vypnutý. To znamená, že v prípade poklesu napätia nie je garantovaný validný stav mikrokontroléra. V prípade použitia tohto režimu musí externá elektronika zabezpečiť, že vstupné

napätie nikdy neklesne pod stanovenú hodnotu, alebo musí byť implementovaný externý obvod na reštartovanie mikrokontroléra.

Z hľadiska zložitosti vývoja, je najjednoduchšie používať jeden z režimov LLS, keďže sa po prebudení mikrokontrolér obnoví do stavu, v akom sa nachádzal pred prechodom do tohto režimu. Režimy Very Low Leakage Stop (VLLS) prinášajú ešte nižšiu spotrebu za cenu dlhšieho prebúdzania mikrokontroléra a zložitejšieho vývoja, keďže mikrokontrolér musí začať vykonávať program od začiatku, podobne, ako po reštartovaní mikrokontroléra [6].



Obrázek 2.7: Podporované prechody medzi režimami prevádzky Kinetis K60. Zdroj: NXP Semiconductors (2015) [7]

2.1.4 Mikrokontrolér ESP32-S3

Mikrokontrolér ESP32-S3 patrí do rodiny 32-bitových mikrokontrolérov založených na architektúre Xtensa LX7. Tento mikrokontrolér je navrhnutý pre moderné aplikácie, ponúka oproti konkurencii vyšší výpočtový výkon, moderné periférie a prepracovaný operačný systém Real-Time Operating System (RTOS) s knižnicami na obsluhu periférií.

Mikrokontrolér ponúka širokú sadu periférií, napríklad Wi-Fi, Bluetooth, a samozrejme štandardné periférie, ako ADC, I2C, Serial Peripheral Interface (SPI) a UART [15]. Veľkou výhodou Embedded System Platform 32-bit (ESP32) je možnosť rozšíriť internú FLASH pamäť o externú, pomocou Octal SPI rozhrania. To je vynikajúca možnosť pre rozsiahlejšie aplikácie využívajúce Wi-Fi, HTTPS a iné protokoly, ktoré vyžadujú viac pamäte, ako typické vstavané aplikácie. Mikrokontrolér podporuje aj pripojenie externej pamäte Pseudo-Static Random Access Memory (PSRAM), ktorá môže byť priamo pripojená pomocou Memory Management Unit (MMU), a aplikácie ju môžu využívať rovnako, ako internú RAM. Vďaka externej RAM nie je aplikačná vrstva prakticky vôbec limitovaná veľkosťou pamäte, čo je v oblasti vstavaných systémov jedinečná vlastnosť, ktorá môže značne ušetriť čas pri vývoji.

Mikrokontrolér ESP32-S3 podporuje aj rozšírené bezpečnostné funkcie secure boot, šifrovanie pamäte FLASH a digitálne podpisovanie binárnych súborov. Pomocou týchto funkcií je možné dosiahnuť vysokú úroveň ochrany pred neoprávneným prístupom a manipuláciou. Výrobca pre mikrokontroléry vytvoril vlastné vývojové prostredie, konfiguračný nástroj, knižnice a upravený operačný systém RTOS. Knižnice k periférium sú prepracované a veľmi podrobné, čo umožňuje vývojárom sústrediť sa na vývoj aplikačnej vrstvy, keďže všetky vrstvy bližšie k hardvéru sú väčšinou už naimplementované a funkčné [15].

Kvôli prepracovaným funkciám, rozsiahlej podpore knižníc a nízkej cene je mikrokontrolér ESP32 veľmi populárny a používa sa v rozličných oblastiach.

Napájacie a hodinové obvody ESP32-S3

Napájacie obvody sú podľa referenčného manuálu [15] rozdelené na viacero domén, čo je dôležité z hľadiska šetrenia energie. V režimoch spánku sa môžu niektoré domény odpojiť, čo zníži spotrebu energie.

ESP32-S3 má tri hlavné napájacie domény. Prvou je digitálna doména (DIG), ktorá napája jadro Xtensa LX7, väčšinu periférií mikrokontroléra a pamäte RAM, SRAM. Túto doménu je možné od napájania odpojiť v režimoch spánku. Zaujímavosťou je, že mikrokontrolér dokáže v režime ľahkého spánku znížiť úroveň napätia v tejto doméne, čím zníži spotrebu energie, ale zachová kontext všetkých periférií a pamätí.

Druhou doménou je Wi-Fi a Bluetooth doména, ktorá napája všetky moduly bezdrôtovej komunikácie. Túto doménu je možné úplne odpojiť v režimoch spánku, čo zníži spotrebu energie na úroveň mikrokontrolérov, ktoré bezdrôtovú komunikáciu nepodporujú.

Poslednou doménou je RTC doména. Tá napája RTC, časovač, časť digitálnych vstupov, ktoré sú pripojené do RTC kontroléra, a koprocesor s nízkou spotrebou energie. Táto doména ostáva napájaná aj v režimoch spánku a umožňuje prebudenie celého systému aj bez aktivity hlavného jadra.

ESP32-S3 má hlavný zdroj hodinového signálu Phase-Locked Loop (PLL), ktorý generuje signál pre procesor a vysokorýchlostné periférie. PLL sa v režimoch spánku vypína, čo znižuje spotrebu.

Druhým zdrojom hodinového signálu je RC oscilátor, ktorý sa využíva v režimoch spánku pre RTC a koprocesor [15].

Režim aktívnej prevádzky mikrokontroléru

Narozdiel od ostatných mikrokontrolérov má ESP32-S3 len jeden režim aktívnej prevádzky. V tomto režime sú napájané všetky periférie. Užívateľ si môže zvoliť z troch frekvencií hodinového signálu (80/160/240 MHz) pre jadro mikrokontroléra. Zaujímavou funkciou, ktorú knižnica ESP-IDF podporuje, je dynamická zmena frekvencie počas aktívnej prevádzky podľa vyťaženia mikrokontroléra. Dynamická zmena je známa z bežných procesorov, ale vo vstavanom svete sa bežne nepoužíva. Táto funkcia môže priniesť zníženie spotreby bez nutných rozsiahlych zásahov do programu.

ESP32-S3 má v aktívnom režime oproti konkurenčným mikrokontrolérom vyššiu spotrebu. Pokiaľ je aktívny modul Wi-Fi, tak spotreba môže dosiahnuť až 240 mA.

Režimy ľahkého spánku

Mikrokontrolér podporuje dva režimy ľahkého spánku. Prvý, tzv. modem sleep, necháva aktívne všetky periférie mikrokontroléra okrem modulu Wi-Fi a Bluetooth. Mikrokontrolér

v tomto režime naďalej vykonáva inštrukcie, modul Wi-Fi je v režime spánku a periodicky sa budí tak, aby bola funkcia Wi-Fi a Bluetooth naďalej zachovaná.

Druhý režim ľahkého spánku vypne hodinový signál pre procesor a väčšinu periférií. Napájanie ostáva aktívne pre všetky periférie a jadro. To umožňuje zachovanie plného kontextu a po prebudení môže mikrokontrolér okamžite pokračovať v práci. V režime ľahkého spánku je aktívny koprocesor s veľmi nízkou spotrebou, modul RTC, časovač RTC a periférie pripojené k modulu RTC. Prebudenie z ľahkého spánku je možné pomocou RTC časovača, hociktorého GPIO vstupu a taktiež periférie UART [14].

Režim hlbokého spánku

V režime hlbokého spánku je hlavná napájacia doména, ktorá napája jadro a všetky periférie, vypnutá. Pri prechode do tohto režimu sa stratí kontext spusteného programu. Mikrokontrolér musí po prebudení znovu prejsť celou inicializáciou rovnako, ako pri prvom zapnutí.

Napájanie pre koprocesor a RTC modul je aktívne, čo umožňuje systému prebudiť pomocou časovača, GPIO vstupov pripojených k RTC modulu a vyvolaním prerušenia koprocesorom.

Prechod do režimu hlbokého spánku umožňuje natívna knižnica [14].

```
esp_deep_sleep_start()
```

Ďalšou funkciou, ktorú mikrokontrolér podporuje, je izolácia digitálnych vstupov a výstupov. Niektoré GPIO majú externe pripojený napájací alebo zemniaci rezistor, čo znamená, že cez nich tečie prúd aj v režimoch spánku. Jednou možnosťou, ako tento prúd obmedziť, je správne nastaviť interný rezistor na každom výstupe. To nemusí vždy stačiť, niekedy je potrebné vykonať zmeny v hardvéri na Printed Circuit Board (PCB). Druhou možnosťou je tieto GPIO izolovať, čo znamená, že sa od zvyšku mikrokontroléra úplne odpoja a nebude cez nich tiecť žiaden prúd. Túto funkciu nemá žiaden konkurenčný mikrokontrolér. Izoláciu GPIO je možné docieliť volaním knižničnej funkcie pred prechodom do hlbokého spánku [14].

```
// Fyzické odpojenie GPIO výstupov od zvyšku mikrokontroléra
rtc_gpio_isolate(GPIO_NUM_12);
```

RTC kontrolér a koprocesor

ESP32-S3 má oddelenú napájaciu doménu pre RTC kontrolér a koprocesor s veľmi nízkou spotrebou, čo z nich robí ideálne periférie na používanie v režimoch spánku. RTC kontrolér poskytuje okrem bežnej funkcie udržiavania času a dátumu aj časovač, pomocou ktorého je možné generovať prerušenie na prebudenie hlavného jadra s presnosťou na mikrosekundy. K RTC kontroléru sú taktiež pripojené vybrané GPIO signály. Týmto signálom je možné nakonfigurovať napájací alebo zemniaci rezistor, ktorý ostane pripojený aj v režime hlbokého spánku. Na každý zo vstupov je možné nakonfigurovať generovanie prerušenia, ktoré prebudí zvyšok mikrokontroléra [14].

```
// Nastavenie vstupných GPIO, ktoré môžu pomocou RTC systém prebudiť
esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
rtc_gpio_pullup_dis(gpio_num);
rtc_gpio_pulldown_en(gpio_num);
```

Mikrokontrolér ESP32-S3 obsahuje koprocesor s veľmi nízkou spotrebou s architektúrou RISC-V. Ten je plne programovateľný v jazyku C a umožňuje vykonávať zložitejšie operácie s perifériami, aj keď je hlavný procesor v režime hlbokého spánku.

Koprocesor má prístup k perifériám v RTC doméne, pamäť zdieľa s RTC kontrolérom. Koprocesor môže ovládať ADC periférium, spúšťať nové merania a vyhodnocovať výsledky. Taktiež môže ovládať GPIO signály, ktoré sú pripojené k RTC kontroléru. Koprocesor taktiež podporuje I2C komunikáciu na týchto GPIO.

Naprogramovaný koprocesor dokáže napríklad počítať pulzy na GPIO vstupoch, vyhodnocovať merania z ADC, alebo udržiavať plnohodnotnú komunikáciu s externými čipmi cez I2C zbernicu a následne prebudiť hlavný procesor, s ktorým môže zdieľať namerané hodnoty alebo dáta z komunikácie.

Všetky funkcie sú dostupné počas toho, ako je zvyšok mikrokontroléra v režime hlbokého spánku, čo umožní systému dosiahnuť bezkonkurenčne najnižšiu spotrebu počas práce s podporovanými periférami. Jedinou nevýhodou je, že programátor musí vyvinúť a udržiavať program pre dva procesory namiesto jedného, čo môže predĺžiť a predražiť vývoj [14].

2.2 Knižnica NXP Power management framework

NXP Power management framework je knižnica pre mikrokontroléry firmy NXP, ktorá zjednodušuje správu spotreby energie mikrokontrolérov. Knižnica abstrahuje architektúru rôznych mikrokontrolérov tak, aby vývojár nemusel program upravovať pre každú konkrétnu architektúru samostatne. Knižnicu je možné použiť spolu s operačným systémom RTOS alebo samostatne. Hlavnou úlohou knižnice je sprostredkovať vývojárovi jednoduché rozhranie, pomocou ktorého môže ovládať jednotlivé režimy prevádzky mikrokontroléru a taktiež dokáže automaticky ovládať spotrebu energie periférií, ktoré sa aktuálne nepoužívajú [7].

2.2.1 Základné funkcie knižnice

Knižnica umožňuje automaticky vypínať všetky prostriedky mikrokontroléru, ktoré sa aktuálne nevyužívajú. Automaticky manažuje prechody medzi stavmi nízkej spotreby energie úpravou registrov mikrokontroléru. Aplikačná vrstva sa môže prihlásiť na prijímanie notifikácií o prechode systému medzi jednotlivými stavmi nízkej spotreby. Programátor môže špecifikovať obmedzenia na rýchlosť prebudenia systému, špecifikovať, ktoré signály majú systém prebudiť z režimu spánku, a knižnica automaticky vyberie, ktorý stav nízkej spotreby energie vyhovuje zvoleným požiadavkám [7].

2.2.2 Aplikačné rozhranie NXP Power management framework

Táto sekcia popisuje z užívateľského hľadiska prácu s knižnicou. Ukazuje, ako sa v knižnici definujú obmedzujúce podmienky, pridávajú signály na prebudenie z režimu spánku a analyzuje obslužné funkcie určené na ovládanie mikrokontroléru. [8].

Knižnica umožňuje užívateľovi definovať rôzne režimy, v ktorých sa systém nachádza. V každom režime je potrebné definovať podmienky, ktoré musia byť splnené pomocou nasledujúcich makier:

```
// Definícia obmedzení pre režim Deep sleep, užívateľ požaduje zachovanie
↪  kontextu SRAM
#define APP`DEEP_SLEEP_CONSTRAINTS 2U, PM_RESC_SRAM16_256KB_RETENTION,
↪  PM_RESC_FLEXSPI0_SRAM_RETENTION
```

V tomto makre je definovaný režim tvrdého spánku, v ktorom užívateľ požaduje, aby ostali uložené dáta v pamäti SRAM a konfigurácia periférie FlexSPI0.

V nasledujúcej ukážke je definovaný režim jednoduchého spánku, v ktorom sú definované požiadavky na zapnuté oscilátory Main Clock, Low power Clock a PLL Clock.

```
// Definícia obmedzení pre režim sleep, užívateľ požaduje aktívne generátory
↪  hodinového signálu
#define APP_SLEEP_CONSTRAINTS 7U, PM_RESC_MAIN_CLK_ON, PM_RESC_LPOSC_ON,
↪  PM_RESC_SYSPLLLDO_ON
```

Dôležitou časťou je definovať vstupné signály, pomocou ktorých sa mikrokontrolér prebudí zo stavu spánku.

```
1 // Povolenie prebudenia pomocou GPIO vstupov
2 PM_InitWakeupSource(&g_UserkeyWakeupSource, (uint32_t)PIN_INT0_IRQn, NULL, true);
```

Funkcia zaregistrovala vstup GPIO ako zdroj prerušenia, pomocou ktorého sa systém prebudí z režimu spánku. Toto rozhranie umožňuje definovať rôzne zdroje prerušení a každý z nich môže systém prebudiť z iného režimu spánku.

API rozhranie umožňuje nastaviť definované požiadavky na zapnuté periférie z ukážky 2.2.2, knižnica automaticky vyberie vhodný režim spánku, ktorý umožňuje ponechať zapnuté všetky špecifikované periférie [8].

```
1 // Nastavenie vopred definovaných obmedzení, knižnica vyberie automaticky vhodný
↪  režim spánku
2 PM_SetConstraints(PM_LP_STATE_NO_CONSTRAINT, APP_DEEP_SLEEP_CONSTRAINTS);
```

Druhou možnosťou je definovať podmienky pre konkrétny režim spánku, čo umožňuje užívateľovi definovať, za akých podmienok sa do daného režimu prejde.

```
1 // // Nastavenie vopred definovaných obmedzení, užívateľ vyberá požadovaný režim
↪  spánku
2 PM_SetConstraints(PM_LP_STATE_DEEP_SLEEP, APP_DEEP_SLEEP_CONSTRAINTS);
```

Pokiaľ nastane nezhoda medzi zvolenými podmienkami a režimom spánku, plánovač automaticky vyberie menej úsporný režim spánku, ktorý vyhovuje všetkým definovaným podmienkam a podporuje ho použitý mikrokontrolér.

Po nakonfigurovaní knižnice je možné prejsť do režimu spánku zavolaním obslužnej funkcie.

```
// Vstup do režimu spánku na daný čas
PM_EnterLowPower(durationTicks);
```

Pomocou parametru funkcie užívateľ definuje na akú dlhú dobu chce prejsť do režimu spánku. Problém je, že prechod z režimu spánku do normálneho režimu trvá každému mikrokontroléru určitú dobu. Pokiaľ by užívateľ zvolil príliš krátky čas v režime spánku, tak by sa z neho mikrokontrolér nestihol zobudiť a celý prechod by bol zbytočný. Doba, za ktorú sa mikrokontrolér prebudí z režimu spánku, sa nazýva latencia prebudenia. Z tohto dôvodu má knižnica pre každý režim spánku a každý mikrokontrolér definovanú latenciu prebudenia. V prípade, že je užívateľom zvolený požadovaný čas v danom režime príliš krátky, knižnica automaticky zvolí režim, z ktorého sa mikrokontrolér prebudí rýchlejšie a splní tak požadovanú podmienku.

Pokiaľ užívateľská aplikácia vyžaduje, aby sa v určitom stave programu mikrokontrolér neuspával, knižnica ponúka jednoduché rozhranie na vypnutie a zapnutie celého manažmentu spotreby [8].

```
// Zapnutie alebo vypnutie knižnice na riadenie spotreby
PM_EnablePowerManager(bool enable);
```

2.3 Zephyr Power management

Zephyr Project je operačný systém pracujúci v reálnom čase s otvoreným zdrojovým kódom. Zephyr naberá na popularite z dôvodu, že podporuje mikrokontroléry od rôznych výrobcov a umožňuje vytvárať prenositeľný kód. Operačný systém ponúka pokročilé možnosti konfigurácie periférie a uľahčuje programátorom vývoj pre rôzne typy mikrokontrolérov. Jednou z častí Zephyr Project je modul Power Management, ktorý prináša aplikačné rozhranie nezávislé na platforme slúžiace na manažment spotreby energie mikrokontroléra. Zephyr Power Management sa skladá z dvoch modulov, systémového manažmentu spotreby a periférneho manažmentu spotreby [10].

2.3.1 Systémový manažment spotreby

Systémový manažment spotreby riadi spotrebu na systémovej úrovni a umožňuje mikrokontroléru prejsť do režimu nízkej spotreby v prípade, že je program v stave nečinnosti. Pokiaľ kernel operačného systému nemá naplánované žiadne procesy čakajúce na výpočet, vypočíta sa doba predpokladanej nečinnosti mikrokontroléru a notifikuje sa manažment spotreby. Ten na základe predpokladaného času nečinnosti vyberie vhodný režim nízkej spotreby energie, do ktorého mikrokontrolér prejde. Aplikačná vrstva má možnosť defino-

vať vstupné signály, ktoré umožnia procesoru prebudiť sa z režimu spánku skôr, ako bol naplánovaný čas prebudenia od operačného systému [17].

Modul definuje režimy nízkej spotreby, ktoré sú implementované ovládačmi jednotlivých mikrokontrolérov. Modul má ku každému režimu priradené, ktoré periférie môžu byť v danom režime aktívne. V operačnom systéme sú definované nasledujúce režimy:

- PM_STATE_ACTIVE — systém je plne aktívny.
- PM_STATE_RUNTIME_IDLE — všetky jadrá procesoru sú v režime najnižšej spotreby a čakajú na prerušenie, ktoré ich prepne do plne aktívneho režimu.
- PM_STATE_SUSPEND_TO_IDLE — všetky jadrá procesoru sú v režime najnižšej spotreby a periférie sú v režime nízkej spotreby. Kontext všetkých jadier procesora je zachovaný.
- PM_STATE_SUSPEND_TO_RAM — Čo najväčšia časť mikrokontoléru je odpojená od napájania, konfigurácia periférii je uložená v pamäti RAM a pri obnovení musí byť znovu načítaná do registrov.
- PM_STATE_SUSPEND_TO_DISK — Všetky periférie vrátane pamäte RAM sú odpojené od napájania, kontext systému je uložený do nevolatilnej pamäte.
- PM_STATE_SOFT_OFF — Mikrokontrolér spotrebúva minimum energie, kontext procesoru nie je zachovaný, po prebudení sa program začína vykonávať od začiatku, podobne ako pri reštarte.

Manažér politiky podľa typu mikrokontroléra rozhoduje, do ktorého režimu systém prejde. Nie každý mikrokontrolér musí podporovať všetky definované režimy. Manažér politiky sa vždy snaží dosiahnuť režim najnižšej spotreby energie. Pri rozhodovaní využíva informácie o potrebnom čase, ktorý mikrokontrolér potrebuje na prechod do režimu nízkej spotreby a prechod naspäť do aktívneho režimu [17].

Aplikačná vrstva môže rozhodovať, do ktorého režimu má mikrokontrolér prejsť pomocou API rozhrania:

```
pm_policy_next_state();
```

Pokiaľ aplikačná vrstva potrebuje čo najrýchlejšiu odozvu systému, môže zamknúť celý modul manažmentu spotreby. Po zavolaní funkcie na zamknutie zostane systém v aktuálnom režime, až kým užívateľ znovu aktivuje systém manažmentu spotreby.

```
pm_policy_state_lock_get()
```

2.3.2 Manažment spotreby energie periférií

Manažment spotreby energie periférií umožňuje perifériám prechádzať do režimu nízkej spotreby energie alebo ich úplne vypnúť v prípade, že sa s perifériou nebude v blízkej dobe pracovať. Systém podporuje dva typy riadenia spotreby energie periférie. Ovládače periférií môžu sami rozhodovať o režime periférie na základe aktuálneho využitia. Druhou možnosťou je, že operačný systém riadi stav periférií podľa aktuálneho režimu systému [17].

Spotreba periférií riadená ovládačmi periférií

V tomto režime riadi režim periférie jej ovládač. K rozhodovaniu, či sa môže periféria prepnúť do režimu spánku, sa používa počítadlo ukazateľov. Pokiaľ počet ukazateľov klesne na nulu, periféria prejde do režimu spánku. Používateľská vrstva môže zakázať periférii prejsť do režimu spánku v prípade, že požaduje rýchlu odozvu.

Hlavná výhoda tohto režimu je, že mikrokontrolér môže rýchlejšie prechádzať do stavov nízkej spotreby a nemusí riadiť periférie, ktoré sa algoritmicky rozhodujú, kedy prejdú do režimu spánku [17].

Spotreba periférií riadená operačným systémom

V prípade, že je spotreba periférií riadená operačným systémom, periférie prechádzajú do režimu spánku spolu s mikrokontrolérom. Keď riadenie systému spotreby vyhodnotí, že mikrokontrolér nemá dlhší čas žiadnu naplánovanú úlohu, tak rozhodne, že má mikrokontrolér prejsť do režimu spánku. Mikrokontrolér najskôr uspí všetky periférie a až potom prejde do režimu spánku. Pri prechode do plne operačného režimu mikrokontrolér po prebudení aktivuje všetky periférie do pohotovostného režimu.

Táto metóda riadenia spotreby periférií je výhodná najmä pre jednoduché systémy s perifériami, ktoré nevyžadujú žiadne blokujúce operácie pri prechode do režimu spánku [17].

2.3.3 Stavy nízkej spotreby periférií

Zephyr knižnica definuje stavy, ktorými sa riadi spotreba každej periférie.

- PM_DEVICE_STATE_ACTIVE — Periféria je aktívna
- PM_DEVICE_STATE_SUSPENDED — Periféria je pozastavená. Kontext periférie môže byť stratený
- PM_DEVICE_STATE_SUSPENDING — Periféria prechádza do režimu pozastavenia.
- PM_DEVICE_STATE_OFF — Periféria má odpojené napájanie. Kontext periférie je stratený.

Implementácia ovládača pre periférium s podporou úspory energie je definovaná pomocou jednotného rozhrania. Počas implementácie stačí do rozhrania doplniť jednotlivé akcie pri prechode do režimov úspory energie.

```c
static int dummy_driver_pm_action(const struct device *dev, enum pm_device_action
↪ action)
{
    switch (action) {
    case PM_DEVICE_ACTION_SUSPEND:
        /* suspend the device */
        ...
        break;
    case PM_DEVICE_ACTION_RESUME:
        /* resume the device */
        ...
        break;
    }
}
```

Prechod do úsporného režimu je možné vynútiť z príkazového riadku. Taktiež je možné zobraziť zoznam všetkých periférií spolu so stavmi, v ktorých sa nachádzajú. V prípade, že periféria vykonáva operáciu, ktorá nesmie byť pozastavená, ovládač môže krátkodobo zakázať riadenie spotreby pre periférium [17].

2.3.4 Definícia úsporných stavov pre systém

Operačný systém Zephyr podporuje definíciu stavov nízkej spotreby energie pomocou konfiguračných súborov. Najskôr je potrebné definovať podporované úsporné režimy pre mikrokontrolér. Pre každý režim sa určuje, či sa pri prechode doň stratí kontext alebo je potrebné jeho zálohovanie. Taktiež je definovaný minimálny čas, ktorý mikrokontrolér potrebuje na prechod do daného režimu a taktiež čas, za ktorý sa mikrokontrolér prebudí [17].

```
power-states {
    state0: state0 {
            compatible = "zephyr,power-state";
            power-state-name = "suspend-to-idle";
            min-residency-us = <10000>;
            exit-latency-us = <100>;
    };

    state1: state1 {
            compatible = "zephyr,power-state";
            power-state-name = "standby";
            min-residency-us = <20000>;
            exit-latency-us = <200>;
    };

    state2: state2 {
            compatible = "zephyr,power-state";
            power-state-name = "suspend-to-ram";
            min-residency-us = <50000>;
            exit-latency-us = <500>;
    };
}
```

Niektoré úsporné režimy procesora môžu odpájať napájanie pre periférie. Na predídenie situácie, kedy sa procesor prepne do úsporného režimu počas prebiehajúcej transakcie na periférii, sa používajú zámky, ktoré periférie zamknú počas transakcie a systém môže prejsť do režimu spánku až po ich odomknutí. Stavy mikrokontroléra, ktoré vypínajú napájanie určitým perifériám, sa taktiež definujú v konfiguračnom súbore [17].

```
test_dev: test_dev {
    compatible = "test-device-pm";
    status = "okay";
    zephyr,disabling-power-states = <&state1 &state2>;
};
```

2.3.5 Zobúdzanie systému externými signálmi

Niektoré periférie majú možnosť prebudiť systém z režimu spánku na základe externých signálov. V konfiguračnom súbore je možné pre jednotlivé periférie povoliť prebúdzanie systému na základe vyvolaných udalostí [17].

```
gpio0: gpio@40022000 {
    compatible = "ti,cc13xx-cc26xx-gpio";
    reg = <0x40022000 0x400>;
    interrupts = <0 0>;
    status = "disabled";
    label = "GPIO_0";
    gpio-controller;
    wakeup-source;
    #gpio-cells = <2>;
};
```

2.3.6 Architektúra knižnice

Knižnica sa skladá z dvoch základných častí. Základné jadro, ktoré je pre všetky mikrokontroléry rovnaké, a aplikovaná časť, ktorá je špecifická pre daný mikrokontrolér.

Základné jadro je tvorené Application Programming Interface (API) rozhraním určeným pre použitie v aplikačnej vrstve. Vďaka API rozhraniu je knižnica prenositeľná medzi rôznymi typmi mikrokontrolérov. Jadro je ďalej tvorené z modulu politiky, v ktorom sú definované všetky podmienky na funkčnosť jednotlivých prvkov systému a algoritmus na výber režimu s najnižšou spotrebou energie, ktorý spĺňa definované podmienky. Ďalší modul definuje všetky vstupné signály, pomocou ktorých je systém možné prebudiť. V aplikačnej vrstve je vďaka tomu možné jednoducho definovať, na základe ktorých signálov požaduje užívateľ prebudenie systému. Posledným modulom je notifikačný modul, ktorý definuje všetky udalosti, ktoré môžu v systéme nastať. Modul ponúka možnosť registrovania notifikácií, ktoré aplikačnej vrstve oznamujú prechody medzi stavmi v systéme.

Aplikovaná časť popisuje pre každý mikrokontrolér sekvencie, pomocou ktorých sa prepína medzi jednotlivými stavmi. Súčasťou aplikovanej časti je aj množina všetkých obmedzení, ktoré pre systém platia. Každá periféria mikrokontroléra má definované, v ktorom stave nízkej spotreby funguje, aký zdroj hodinového signálu vyžaduje a či je možné pomocou tejto periférie systém zobudiť zo stavu nízkej spotreby energie. Aplikovaná časť vychádza z referenčného manuálu pre daný čip, v ktorom sú definované všetky obmedzenia a funkcie [17].

2.4 Meranie spotreby mikrokontrolérov

Meranie spotreby a porovnávanie jednotlivých mikrokontrolérov je náročné, keďže spotreba závisí od viacerých premenných. Mikrokontrolér väčšinou nezostáva v jednom režime prevádzky, tie sa dynamicky prepínajú podľa toho, aká úloha je práve vykonávaná. Ďalším dôležitým faktorom je požadovaný výpočtový výkon pre konkrétnu úlohu. Mikrokontrolér, ktorý bude mať zníženú frekvenciu hodinového signálu, bude mať nižšiu spotrebu, ale jeho výpočtový výkon nemusí pre danú úlohu stačiť. Spotreba taktiež záleží od použitých periférií, frekvencie využitia a schopnosti periférií prejsť do režimu spánku v prípade nečinnosti. Spotreba je taktiež ovplyvnená optimalizáciou softvéru a od toho, ako často softvér umožní mikrokontroléru prejsť do režimu spánku. Niektorí výrobcovia ponúkajú aplikácie, ktoré dokážu vypočítať odhadovanú spotrebu mikrokontroléra na základe zadaných vstupných parametrov. Faktom ale je, že počas výberu mikrokontroléra ešte nemusí byť jasná

výpočtová náročnosť aplikácie a ani požiadavky na funkciu jednotlivých periférií. Jednou z možností je použiť výsledky zo štandardizovaných testov, ktoré ponúkajú niektoré firmy. Všetky mikrokontroléry vykonávajú štandardizovanú úlohu a výsledky ich spotreby je tak možné priamo porovnávať [4].

2.4.1 Testy spotreby ULPMark

Embedded Microprocessor Benchmark Consortium (EEMBC) sa špecializuje na vývoj štandardizovaných testov pre vstavané zariadenia. Poskytujú rôzne testy výkonnosti, rýchlosti komunikácie, bezpečnosti a taktiež testy spotreby mikrokontrolérov. Na ich stránkach sú zverejnené výsledky všetkých testov pre vybrané mikrokontroléry, a taktiež ponúkajú sady testov, ktoré môže používateľ implementovať do svojej aplikácie a výsledky porovnať s dostupnou databázou. Testová sada ULPMark sa sústredí na testovanie spotreby mikrokontrolérov v kombinovaných zaťaženiach. [2]

Test ULPMark Core

Test spotreby ULPMark Core sa zameriava na odmeranie spotreby jadra mikrokontroléra. Počas testu je mikrokontrolér určitý čas aktívny a musí vykonať nasledujúce úlohy:

- Vygenerovať 20 impulzov na výstupoch mikrokontroléra
- Vypočítať 8 bitovú lineárnu interpoláciu
- Vypočítať 16 bitovú integráciu hodnôt
- Zobraziť hodnotu na 7 segmentovom LCD displeji
- Vyhladať podreťazec v reťazci
- Vykonať usporiadanie poľa pomocou bubble-sort algoritmu
- Bitové permutácie reťazca

Po vykonaní všetkých úloh môže mikrokontrolér prejsť do režimu spánku až do kým nie je potrebné vykonať ďalšiu sériu úloh. Výber režimu závisí od podporovaných funkcií mikrokontroléra. Je potrebné vybrať režim s najnižšou spotrebou, ale musí sa zvážiť aj čas prechodu späť do aktívneho stavu a prípadne obnovenie stavu pamäte RAM a obsahu registrov periférií, keďže sa aj tieto úkony počítajú do celkovej spotreby [3].

Test ULPMark Peripheral

Kým ULPMark Core testoval hlavne spotrebu jadra mikrokontroléra, ULPMark peripheral sa zameriava na test spotreby periférií. V súbore testov sa využívajú bežne používané periférie mikrokontroléra, ako ADC, Pulse Width Modulation (PWM), SPI a RTC. Test zahŕňa desať aktivít, z ktorých sa každá vykonáva jednu sekundu. Sekvencia aktivít je popísaná v tabuľke A.1. Spotreba závisí od jednotlivých mikrokontrolérov a funkcií ich periférií. Pokiaľ má mikrokontrolér rýchlejšie periférie, môže rýchlejšie vykonať jednotlivé úlohy a následne periférie a jadro prepnúť do režimu spánku [3].

Kapitola 3

Návrh knižnice na riadenie spotreby mikrokontrolérov

Hlavnou motiváciou pre vývoj knižnice na riadenie spotreby mikrokontrolérov je poskytnúť užívateľom abstraktnú vrstvu, ktorá bude jednoduchá na pochopenie a použitie. Každý výrobca mikrokontrolérov má implementované režimy šetrenia energie inak a dokonca sa líšia aj medzi jednotlivými modelmi mikrokontrolérov. Abstraktná knižnica by mala poskytovať jednotné rozhranie, pomocou ktorého by bolo možné ovládať režimy spánku mikrokontrolérov od rôznych výrobcov. Mala by poskytovať štandardizované režimy spánku, ktoré sa následne v ovládači prispôsobia funkciám, ktoré podporuje mikrokontrolér. Z každého režimu by malo byť jasné, ktoré pamäte a periférie je potrebné obnoviť po prebudení a ktoré ostanú zachované. Knižnica by mala poskytovať jednoduchý systém, ktorý umožní jednotlivým perifériám zakázať prechod do režimu spánku v prípade, že daná periféria aktuálne vykonáva kritickú činnosť. Knižnica by taktiež mala poskytovať podporu pre štandardizované prerušenia, ktoré systém môžu prebudiť z režimu spánku. Každé prerušenie sa musí dať aktivovať alebo zakázať pred prechodom do spánku.

Základná verzia knižnice by mala obsahovať implementáciu pre aspoň dva mikrokontroléry od rôznych výrobcov. Na základe tejto implementácie budú mať užívatelia možnosť rozšíriť podporu pre ďalšie mikrokontroléry. Rozhranie ovládačov musí byť jasne definované a štandardizované, aby bolo v budúcnosti jednoduché rozširovať zoznam podporovaných mikrokontrolérov. Implementácia pre každý mikrokontrolér by mala obsahovať definované závislosti pre periférie, ktoré môžu ostať aktívne v jednotlivých režimoch spánku, a taktiež ku každému režimu priradiť všetky podporované prerušenia, ktoré môžu mikrokontrolér prebudiť. V neposlednom rade musí implementácia obsahovať informáciu o obsahu pamätí, ktoré je potrebné po prebudení mikrokontroléra obnoviť. Každý ovládač musí mať definovanú priemernú dobu trvania prebudenia z jednotlivých úsporných režimov, aby sa užívateľ mohol jednoducho rozhodnúť, ktorý použiť. Vďaka tomu bude možné knižnicu v budúcnosti rozšíriť o modul, ktorý bude automaticky vyberať optimálny úsporný režim vzhľadom na očakávanú dĺžku spánku. Knižnica by taktiež mala poskytovať modul, ktorý dokáže vygenerovať základnú štatistiku o tom, koľko času mikrokontrolér strávil v jednotlivých režimoch, čo môže užívateľovi pomôcť pri vývoji aplikácie.

Jedným z hlavných cieľov knižnice je zjednodušiť implementáciu šetrenia spotreby energie v systémoch, kde sú použité mikrokontroléry od rôznych výrobcov. Počas používania knižnice by malo byť jasné, čo každý z režimov prevádzky mikrokontroléra robí, aj bez toho, aby musel užívateľ zdĺhavo študovať referenčné manuály. V neposlednom rade musí byť pre

užívateľa jasné, ktorý režim spánku môže použiť, vzhľadom na aktuálne používané periférie a požiadavky na prebudenie pomocou týchto periférií.

3.1 Analýza požiadaviek

Jedným z požiadaviek na knižnicu je podpora rôznych typov mikrokontrolérov od rôznych výrobcov. Každý typ mikrokontroléra obsahuje iný hardvér na riadenie spotreby a podporuje odlišné funkcie na prechod do režimov spánku. Knižnica preto musí obsahovať vrstvu, ktorá štandardizuje funkcie rôznych typov mikrokontrolérov do jednotného abstraktného rozhrania, ktoré bude rovnaké pre všetky mikrokontroléry. Z toho vyplýva, že knižnica potrebuje vrstvu, v ktorej bude špecifická implementácia funkcií pre konkrétny mikrokontrolér.

V knižnici musí byť možnosť uchovať zoznam všetkých režimov spánku, ktoré mikrokontrolér podporuje. Ku každému režimu spánku by malo byť možné uložiť, ako dlho bude trvať prebudenie z daného režimu a priradiť obmedzujúce podmienky pre jednotlivé periférie. Medzi obmedzujúce podmienky patrí zoznam periférií, ktoré môžu byť v jednotlivých režimoch spánku aktívne, a informácia, či si periféria zachová obsah registrov po prebudení.

Ak budú pre daný mikrokontrolér a jeho režimy spánku v knižnici dostupné obmedzujúce podmienky, tak by malo byť možné definovať heuristiku, podľa ktorej sa rozhoduje, ktorý režim spánku sa použije. Heuristika môže využívať aj informácie o spotrebe mikrokontroléra v každom režime spánku a latenciu prebudenia, aby bolo možné vybrať vhodný režim spánku.

Hardvérová vrstva, ktorá je špecifická pre každý mikrokontrolér, by mala implementovať funkcie na ovládanie jeho spotreby energie. Medzi tieto funkcie patria prechody do jednotlivých režimov spánku a následné obnovenie funkcie mikrokontroléra po prebudení. Taktiež by mala podporovať ovládanie časovačov, prípadne RTC, ktoré dokážu systém prebudiť po uplynutom čase.

Ďalšou požiadavkou je možnosť informovať užívateľa o udalostiach, ktoré prebudili systém z režimu spánku. Po prebudení by knižnica mala znovu inicializovať všetky hardvérové časti mikrokontroléra, ktoré si neuložili svoj kontext. Užívateľ by mal mať možnosť registrovať si funkcie, ktoré sa budú volať v prípade, že nastala v systéme nejaká zmena, ako napríklad prebudenie.

Abstraktná vrstva knižnice by mala obsahovať jednotné rozhranie, ktorým bude možné ovládať režimy spánku mikrokontrolérov nezávisle od hardvéru mikrokontroléra. Z toho vyplýva, že by knižnica mala mať možnosť automaticky vybrať vhodný režim spánku pre MCU bez toho, aby musel používateľ dopodrobna poznať hardvér, s ktorým pracuje.

Knižnica by mala poskytovať možnosti konfigurácie, vypínanie funkcií, ktoré užívateľ nepotrebuje, tak, aby knižnica zaberala čo najmenej miesta v pamäti RAM a FLASH. To umožní využívať knižnicu aj v mikrokontroléroch, ktoré majú obmedzené hardvérové prostriedky. Užívateľ by mal mať možnosť používať hardvérovú vrstvu knižnice samostatne, aby mohol priamo nastavovať režimy nízkej spotreby mikrokontroléra bez použitia abstraktnej vrstvy.

Z analýzy dostupných knižníc vyplýva požiadavka na prepracovaný systém udalostí a upozornení na udalosti. V knižnici by malo byť možné uchovávať informácie o udalostiach, ktoré v systéme nastali. Užívateľ by mal mať možnosť získať informácie o jednotlivých udalostiach, prípadne si registrovať funkcie, pomocou ktorých bude notifikovaný o novej udalosti.

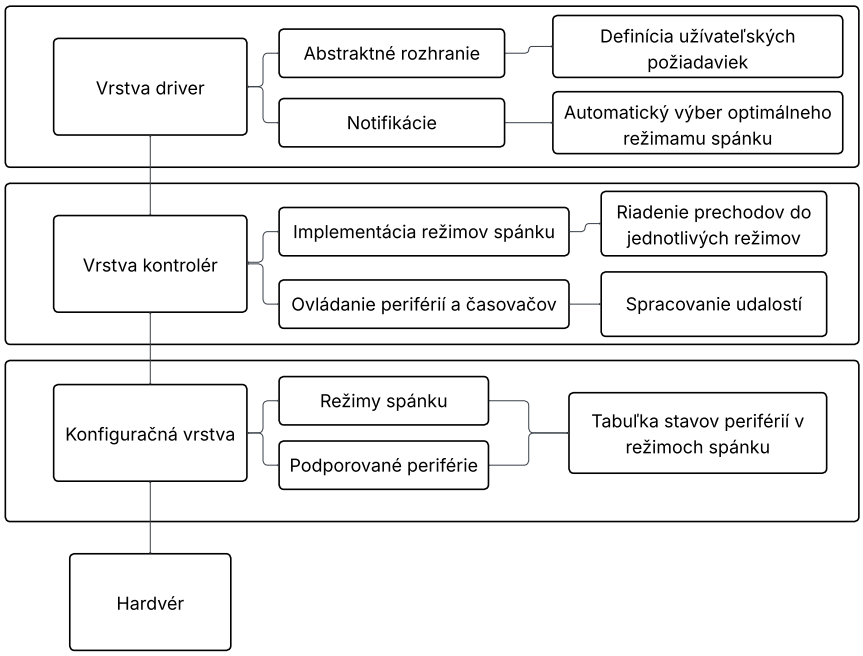
3.2 Architektúra knižnice

Jedným z cieľov tvorby knižnice je použiť modulárny dizajn tak, aby bolo možné do knižnice implementovať podporu pre rôzne typy mikrokontrolérov. Z toho vyplývajú požiadavky na architektúru knižnice. Knižnica by mala mať oddelenú vrstvu, ktorá bude implementovať funkcie pre konkrétny hardvér. Súčasťou tejto vrstvy je konfiguračná vrstva, ktorá ukladá všetky informácie o podporovaných stavoch mikrokontroléra a jeho periférií. Jednotlivé vrstvy knižnice a ich základné úlohy sú popísané na diagrame v obrázku 3.1. Architektúra knižnice vychádza zo šablóny LPM vytvorenej na Fakulte informatiky VUT [13].

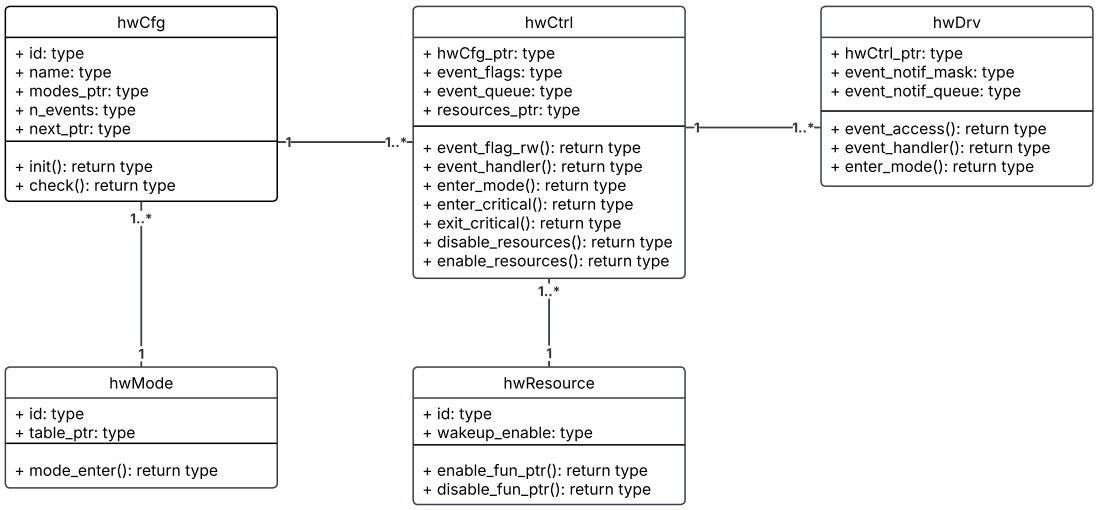
Najnižšou vrstvou knižnice je konfiguračná vrstva, ktorá obsahuje informácie o hardvérovej konfigurácii daného mikrokontroléra. Informácie v tejto vrstve pochádzajú hlavne z referenčného manuálu a popisujú, aké režimy spánku mikrokontrolér podporuje a aké sú obmedzujúce podmienky pre ich použitie.

Druhou vrstvou je vrstva kontrolér, ktorá je zodpovedná za ovládanie hardvérových prvkov mikrokontroléra. Táto vrstva obsahuje obslužné funkcie na konfiguráciu mikrokontroléra, prechod medzi jednotlivými režimami spánku a nastavenie prebúdzania. Táto vrstva priamo pracuje s konfiguračnou vrstvou, keďže z nej získava potrebné informácie o možnostiach daného mikrokontroléra. Súčasťou vrstvy je aj spracovanie udalostí, ktoré v mikrokontroléroch nastali.

Poslednou a zároveň najvyššou vrstvou je vrstva driver. Poskytuje abstraktné rozhranie pre ovládanie spotreby mikrokontrolérov bez potreby poznať možnosti hardvéru. Vrstva využíva unifikované rozhranie z kontroléra, vďaka ktorému dokáže každý mikrokontrolér ovládať rovnakým spôsobom.



Obrázek 3.1: Diagram vrstiev knižnice a ich hlavné úlohy.



Obrázek 3.2: UML Diagram objektov v knižnici.

Konfiguračná vrstva

Konfiguračná vrstva popisuje konfiguráciu konkrétneho hardvéru, pre ktorý je určená. Pre každý podporovaný mikrokontrolér musí byť definovaná konfigurácia v konfiguračnej vrstve.

Konfiguračná vrstva obsahuje v prvom rade identifikačné číslo mikrokontroléra, pomocou ktorého je možné v rámci knižnice jednoducho rozoznať, s ktorým typom mikrokontroléra sa pracuje. Pre mikrokontrolér sú taktiež definované všetky podporované režimy spánku a frekvencie hlavného generátoru hodinového signálu. Ku každému režimu spánku je možné priradiť čas potrebný na prebudenie z daného stavu. Táto informácia pomáha kontroléru pri rozhodovaní, do ktorého režimu spánku môže mikrokontrolér prejsť.

Súčasťou vrstvy je zoznam tzv. udalostí, ktoré môžu v systéme nastať. Udalosti sú väčšinou zhodné s prerušeniami, ktoré daný mikrokontrolér podporuje. Zoznam udalostí sa využíva hlavne pre obsluhu prerušení, ktoré spracúva vrstva kontrolér a taktiež pre používateľa, ktorý pomocou udalostí môže zistiť, čo systém z režimu spánku prebudilo.

Ďalšou dôležitou súčasťou tejto vrstvy je zoznam periférií mikrokontroléra. Pomocou zoznamu periférií je definovaná tabuľka stavov periférií. Tá vyjadruje pre každú periférium, v akom stave sa nachádza, keď sa mikrokontrolér prepne do jedného z režimov spánku. Stav periférie môže byť aktívny, spánok s uchovaním kontextu a úplné vypnutie periférie. Kontrolér následne podľa tejto tabuľky rozhoduje, čo s každou perifériou musí vykonať po prebudení z režimov spánku. Tabuľku je taktiež možné využiť pri rozhodovaní, do ktorého režimu spánku môže mikrokontrolér prejsť na základe potrieb aplikačnej vrstvy. Tá definuje, aké periférie musia ostať v režime spánku aktívne. Zoznam atribútov konfiguračnej vrstvy je napísaný v diagrame tried na obrázku 3.2.

Vrstva kontrolér

Vrstva kontrolér implementuje funkcie na riadenie spotreby mikrokontroléra. Taktiež obsahuje konfiguráciu pre mikrokontrolér, ktorý sa aktívne využíva. Vzťahy medzi vrstvami sú vyznačené na obrázku 3.2.

Vrstva sa rozdeľuje na obecné rozhranie a konkrétnu implementáciu pre daný mikrokontrolér. Obecné rozhranie zahŕňa definíciu udalostí, ktoré môžu nastať. Ku každej udalosti je možné uložiť časovú známku, kedy udalosť naposledy nastala, a nastaviť jej prioritu. Užívateľ môže pre každú udalosť registrovať funkciu, ktorá sa zavolá, keď udalosť nastane.

Kontrolér taktiež definuje ovládanie periférií. Pre každú podporovanú periférium si môže používateľ registrovať funkcie na inicializáciu a odpojenie danej periférie. Tie budú automaticky volané kontrolérom v prípade, že procesor prešiel do režimu spánku alebo z režimu spánku, v ktorom sa stratil kontext periférií. Tento spôsob poskytuje najväčšiu voľnosť pre používateľa, pretože si môže každú periférium nakonfigurovať podľa svojich potrieb a nie je závislý od implementácie ovládača danej periférie z knižnice. Používateľ môže perifériám následne nastaviť, či musia byť aktívne pri prechode do režimu spánku. Podľa toho následne kontrolér vyberie režim spánku, ktorý vyhovuje zvoleným požiadavkám. Trieda periférií je so svojimi atribútmi znázornená v obrázku 3.2.

Súčasťou tejto vrstvy sú všetky obslužné funkcie pre daný hardvér, ktoré sú implementované v časti špecifickej pre každý mikrokontrolér. Ovládač pre každý mikrokontrolér implementuje funkcie na prechod medzi jednotlivými režimami spánku, funkciu na obnovenie mikrokontroléra po prebudení a funkciu na spracovanie prerušení. Súčasťou je aj heuristika, ktorá na základe definovaných obmedzení a požadovaného času zotrvania v režime spánku, vyberie optimálny režim z hľadiska spotreby.

Výhodou tejto architektúry je, že užívateľ môže využívať iba obslužné funkcie implementované pre daný hardvér bez zvyšku knižnice. To mu umožní úplnú kontrolu nad hardvérom bez toho, aby musel manuálne nastavovať registre mikrokontroléra. Alebo môže použiť všetky vrstvy knižnice a využívať tak plne abstraktné rozhranie knižnice.

Ku každej definovanej udalosti je možné v kontroléri registrovať notifikácie, ktoré slúžia na upozornenie užívateľa, že daná udalosť nastala. Každá notifikácia môže mať zaregistrovanú užívateľskú funkciu, ktorá sa zavolá, keď je daná notifikácia vygenerovaná.

Vrstva driver

Táto vrstva je najvyššou vrstvou v knižnici. Je úplne oddelená od hardvéru mikrokontrolérov. Vrstva využíva obslužné funkcie kontroléra na ovládanie procesora a jeho periférií. Obsahuje taktiež funkcie slúžiace na prechod mikrokontroléra do režimu spánku. Užívateľ si nemusí zvoliť konkrétny režim spánku, do ktorého chce prejsť. Stačí zadať dobu, po ktorú chce v spánku zotrvať, a kontrolér automaticky pomocou heuristiky a obmedzujúcich podmienok vyberie vhodný úsporný režim. Konkrétne atribúty vrstvy driver sú popísané na obrázku 3.2.

Vrstva driver taktiež poskytuje štatistické dáta o tom, koľko času mikrokontrolér strávil v režime spánku a počítadlá počtu prebudení pre každú udalosť, ktorá mikrokontrolér môže prebudiť. Z týchto dát je možné vygenerovať štatistiky o tom, ktoré režimy spánku sú najviac používané a užívateľ môže aplikačnú vrstvu upraviť tak, aby mikrokontrolér mohol prejsť do režimu spánku na dlhšiu dobu a ušetriť tak viac energie.

Vrstvu driver je možné v budúcnosti rozširovať o nové funkcie nezávisle od mikrokontrolérov, ktoré knižnica podporuje, keďže je od hardvéru oddelená. To je výhoda aj počas vývoja knižnice, viacero vývojárov môže naraz pracovať na rôznych vrstvách, čo môže vývoj knižnice zrýchliť.

3.3 Aplikačné rozhranie knižnice

Aplikačné rozhranie knižnice je dôležitou súčasťou implementácie. Malo by byť navrhnuté tak, aby bolo užívateľovi na prvý pohľad jasné, čo jednotlivé funkcie robia bez toho, aby musel zdĺhavo študovať dokumentáciu. Knižnica je navrhnutá tak, aby sa nižšie vrstvy knižnice mohli používať samostatne. Pre každú vrstvu musí byť definované aplikačné rozhranie, ktoré bude fungovať nezávisle od vyšších vrstiev a poskytne užívateľom najväčšiu flexibilitu počas používania knižnice. Aplikačné rozhranie je navrhnuté podľa diagramu na obrázku 3.2. Rozhranie knižnice vychádza zo šablóny knižnice Library for Power Management (LPM) [13], ktoré som v rámci diplomovej práce rozšíril o nové funkcie.

3.3.1 Rozhranie konfiguračnej vrstvy

Konfiguračná vrstva popisuje, ako má konkrétny mikrokontrolér implementované režimy spánku a taktiež, ktoré periférie je možné používať v jednotlivých režimoch.

Pre každý mikrokontrolér sú definované názvy režimov nízkej spotreby, ktoré podporuje.

```
typedef enum PowerMode {
    POWER_MODE_RUN    = 0,  /**< Bežný režim (Run) */
    POWER_MODE_SLEEP  = 1,  /**< Režim spánku (Sleep) */
    POWER_MODE_STOP   = 2,  /**< Stop režim */
    POWER_MODE_OFF    = 3   /**< Vypnutý režim (Shutdown) */
} PowerMode_t;
```

Nasleduje definícia všetkých udalostí, ktoré môžu nastať. Typicky sa medzi udalosťami definujú všetky prerušenia, ktoré môžu mikrokontrolér prebudiť, a taktiež udalosti, ako prechod do režimu nízkej spotreby a návrat do režimu aktívnej prevádzky.

```
typedef enum /**< udalosti ktoré podporuje mikrokontrolér */
{
    LPM_HW_EVENT_FAILURE,
    LPM_HW_EVENT_MODE_ENTRY,
    LPM_HW_EVENT_MODE_EXIT,
    LPM_HW_EVENT_UART,
    LPM_HW_EVENT_GPIO,
    LPM_HW_EVENT_RTC,
    LPM_HW_EVENT_I2C
    LPM_HW_EVENT_CNT
} lpmExampleEvent_t;
```

Taktiež je potrebné definovať všetky periférie, ktoré mikrokontrolér obsahuje. Ako periféria sa považuje aj samotné jadro, hlavný obvod hodinového signálu, pamäť RAM a FLASH a napájacie domény. Vďaka tomu je možné pracovať s týmito časťami mikrokontroléra rovnako, ako s perifériami. Pre každý režim nízkej spotreby je možné definovať, či sú dané časti mikrokontroléra aktívne a či sa uchováva ich kontext.

```c
typedef enum
{
    LPM_RESOURCE_CPU,       /**< CPU */
    LPM_RESOURCE_BUSCLK,    /**< Systémová zbernica (Bus clock) */
    LPM_RESOURCE_VDD,       /**< Napájanie VDD */
    LPM_RESOURCE_RAM,       /**< RAM pamäť */
    LPM_RESOURCE_FLASH,     /**< Flash pamäť */
    LPM_RESOURCE_UART,      /**< UART periféria */
    ...
} LpmResource_t;
```

Pre každú periférium sú v konfigurácii definované stavy, v ktorých sa môže nachádzať. Môže byť aktívna, mať vypnutý hodinový signál, zachovať si kontext, odpojené napájanie a nezachovať si kontext. Tieto stavy sa následne priradia k jednotlivým režimom šetrenia energie mikrokontroléra.

```c
typedef enum
{
    LPM_RESOURCE_OFF,    /**< Vypnutý (napájanie/hodiny/retencia sú off) */
    LPM_RESOURCE_ON,     /**< Zapnutý (plne použiteľný bez obmedzení) */
    LPM_RESOURCE_RET,    /**< Retencia (stav je zachovaný) */
    LPM_RESOURCE_OPT,    /**< Voliteľný režim (zapnutý/vypnutý/retencovaný) */
    LPM_RESOURCE_LP      /**< Nízkoenergetický režim (low-power) */
} LpmResource_t;
```

Aby bolo možné definovať pre každý režim úspory energie mikrokontroléra a každú periférium, v akom stave sa periféria nachádza, je v konfiguračnej vrstve definovaná tabuľka. Stĺpce popisujú režimy nízkej spotreby mikrokontroléra a riadky definujú periférie. V každej bunke je vložený jeden stav, v ktorom sa konkrétna periféria nachádza, keď je mikrokontrolér v režime spánku. Táto tabuľka sa využíva pri automatickom výbere režimu spánku, do ktorého mikrokontrolér môže prejsť, vzhľadom na využívané periférie.

```c
static const LpmModeTableCell_t lpmModeTable[LPM_RESOURCE_COUNT][LPM_MODE_COUNT]
↪  = {
  /*                 |  Beh (RUN)          Čakanie (WAIT)     Stop (STOP) */
  /* Flash    */  { LPM_RESOURCE_ON,    LPM_RESOURCE_OPT,   LPM_RESOURCE_OFF },
  /* UART     */  { LPM_RESOURCE_ON,    LPM_RESOURCE_ON,    LPM_RESOURCE_OFF },
  /* SPI      */  { LPM_RESOURCE_ON,    LPM_RESOURCE_ON,    LPM_RESOURCE_OFF },
  /* I2C      */  { LPM_RESOURCE_ON,    LPM_RESOURCE_ON,    LPM_RESOURCE_OFF }
};
```

3.3.2 Rozhranie kontroléra

Vrstva kontrolér je rozdelená na dve časti. Obecnú časť tvoria obslužné funkcie, ktoré sú rovnaké pre všetky mikrokontroléry. Jedná sa o funkcie určené na registráciu používaných

periférií, získavanie informácií o udalostiach, ktoré nastali, a funkciu na prechod mikrokontroléra do režimu spánku. Druhú časť tvorí rozhranie pre ovládanie hardvérových modulov mikrokontroléra. Ovládač musí pre každý mikrokontrolér implementovať funkcie z tohto rozhrania tak, aby ho kontrolér mohol pomocou nich riadiť.

Každý ovládač musí pre konkrétny mikrokontrolér implementovať funkcie na inicializáciu ovládača, identifikáciu hardvéru, funkciu na vstup do režimu nízkej spotreby energie a funkciu na spracovanie prerušení.

```
// Inicializácia kontroléra pre dané MCU
int lpm_hwctrl_init(struct LpmHwCtrl_s* ctrl);

// Kontrola podpory kontroléra pre daný hardvér.
int lpm_hwctrl_check(uint32_t hwId);

// Vstup do režimu spánku na danú dĺžku.
int lpm_hwctrl_enter_mode(struct LpmHwCtrl_s* ctrl, uint32_t milliseconds);

// Spracovanie udalostí hardvéru.
int lpm_hwctrl_event_handler(int eventId);
```

Ovládač môže mať následne definované ďalšie obslužné funkcie, ktoré môže používateľ využívať samostatne, bez zvyšku knižnice. Môže sa jednať napríklad o funkcie na spustenie RTC časovača a vstup do jednotlivých režimov nízkej spotreby. Táto situácia môže nastať v prípade, keď sa používateľ rozhodne použiť len najnižšie vrstvy knižnice a požaduje plnú kontrolu nad riadením spotreby.

Pre každý ovládač je definovaná štruktúra, ktorá obsahuje identifikátor mikrokontroléra, ku ktorému patrí, a ukazatele na funkcie naimplementované v ovládači. Túto štruktúru využíva obecná časť kontroléra.

```
static struct LpmHwCfg_s sLpmHwCfg =
{
    LPM_HW_ID_GENERIC,          /**< ID hardvéru */
#if LPM_USE_HWNAMES == YES
    "lpm generic",              /**< Názov hardvéru */
#endif
    lpm_hwctrl_init,            /**< Funkcia na inicializáciu hardvéru */
    lpm_hwctrl_check,           /**< Funkcia na overenie identity hardvéru */
    lpm_hwctrl_enter_mode,      /**< Funkcia na vstup do konkrétneho režimu
    ↪  hardvéru */
    lpm_hwctrl_event_handler,   /**< Funkcia na spracovanie udalostí hardvéru */
    &sLpmModeCfg,               /**< Konfigurácia režimov hardvéru */
    LPM_EVENT_COUNT,            /**< Počet udalostí podporovaných hardvérom */
    NULL                        /**< Ukazovateľ na ďalší hardvér */
};
```

Základnou časťou obecného rozhrania sú funkcie určené na registráciu hardvéru, ktorý chce užívateľ použiť, a vytvorenie kontroléra.

```
// Registrácia hardvéru.
uint32_t lpm_hw_register(struct LpmHwCfg_s* hwCfg_p);

// Nájde hardvér podľa ID.
struct LpmHwCfg_s* lpm_hw_find(uint32_t hwId);

// Vytvorenie riadiacej štruktúry pre kontrolér hardvéru.
struct LpmHwCtrl_s* lpm_hw_ctrl_create(struct LpmHwCfg_s* hwCfg_p);
```

3.3.3 Rozhranie driveru

Vrstva driveru slúži hlavne ako rozhranie na prácu s knižnicou pre používateľa. Umožňuje prepojiť vrstvy závislé na hardvéri s unifikovaným užívateľským rozhraním. Jednotlivé vrstvy a ich úlohy sú znázornené na obrázku 3.1. Cieľom tejto vrstvy je prepínať mikrokontrolér do režimu nízkej spotreby a registrovať periférie, s ktorými používateľ pracuje.

Na inicializáciu driveru slúžia dve funkcie, ktorým je potrebné ako parameter vložiť dopredu vytvorený kontrolér s registrovaným identifikačným číslom použitého mikrokontroléra.

```
// Inicializácia ovládača v
struct LpmHwDriver_s* lpm_hw_driver_create(struct LpmHwCfg_s* hwCfg_p);
int lpm_hw_driver_init(struct LpmHwDriver_s* drv_p);
```

Súčasťou vrstvy sú funkcie na prácu s perifériami mikrokontroléra. Funkcie umožňujú registrovať periférie, ktoré používateľ využíva. Ku každej periférii si môže používateľ definovať funkciu na inicializáciu a odpojenie periférie. Tieto funkcie sa automaticky zavolajú, keď kontrolér vyhodnotí, že je pred prechodom do režimu spánku potrebné periférium vypnúť z dôvodu šetrenia energie, alebo periférium znovu inicializovať po strate kontextu. Sekvencia prechodu do režimu spánku s obsluhou periférií je znázornená v sekvenčnom diagrame v prílohe C. Toto riešenie poskytuje užívateľovi voľnosť v tom, či bude používať pre svoje periférie oficiálne HAL ovládače, alebo vlastnú proprietárnu knižnicu. V oboch prípadoch bude môcť využiť funkcie knižnice na riadenie spotreby. Pre každú periférium môže užívateľ pred prechodom do režimu spánku vybrať, či vyžaduje podporu prebudenia danou perifériou.

```
// Nájde periférium podľa ID
struct LpmHwResource_s* lpm_hw_ctrl_resource_find(struct LpmHwCtrl_s* ctrl_p,
↪  size_t id);

// Registrovanie hardvérovej periférie s ukazateľmi na funkcie pre zapnutie a
↪  vypnutie
uint32_t lpm_hw_ctrl_resource_register(struct LpmHwCtrl_s* ctrl_p, size_t id,
↪  void (*enable)(void), void (*disable)(void));

// Povolenie/zakázanie prebudenia pomocou hardvérovej periférie.
void lpm_hw_ctrl_resource_enable_wakeup(struct LpmHwCtrl_s* ctrl_p, size_t id,
↪  bool enable);
```

Udalosti v knižnici slúžia na prebudenie systému a informovanie používateľa. Udalosť môže byť napríklad prebudenie systému z režimu spánku pomocou prerušenia vygenerovaného periférium. Užívateľ môže pomocou obslužných funkcií zistiť, ktorá z udalostí nastala a získať o nej podrobnejšie informácie.

```
// Čítanie masky udalosti ovládača knižnice.
bool lpm_hw_driver_read_event_mask(struct LpmHwDriver_s* drv_p, lpmHwEvent_t e);

// Zápis hodnoty do masky udalosti ovládača knižnice.
void lpm_hw_driver_write_event_mask(struct LpmHwDriver_s* drv_p, lpmHwEvent_t e,
↪  bool val);
```

3.4 Proces prechodu do režimu spánku a standby

Na to, aby mikrokontrolér úspešne prešiel do režimu spánku, a následne sa z neho prebudil, je nutné správne nastaviť hardvérové prvky. Tento proces je špecifický pre platformu, na ktorej sa vykonáva, ale v rámci návrhu knižnice ho je potrebné zovšeobecniť na postupnosť krokov, ktorú je možné uplatniť na všetkých platformách. Z tohto hľadiska som definoval dva typy procesu prechodu do režimu spánku, prvý typ popisuje situáciu, kedy si mikrokontrolér zachoval kontext vykonávaného programu. Druhý typ definuje proces, kedy sa kontext programu nezachoval, a mikrokontrolér sa po prebudení reštartuje.

Proces prechodu do režimu spánku a následného prebudenia v prípade, že je zachovaný kontext programu, je znázornený v diagrame C.1. Počas behu programu môže užívateľ požiadať o prechod do vybraného režimu spánku. Knižnica najskôr vypne všetky periférie, ktoré užívateľ nepotrebuje na prebudenie systému. Tieto periférie si môže zvoliť pomocou aplikačného rozhrania. Ďalším krokom je nastavenie časovača, aby sa systém prebudil v požadovanom čase. Prebudenie pomocou časovača sa využije iba v prípade, že počas režimu spánku nebolo vygenerované iné prerušenie. Po prebudení z niektorých režimov spánku je potrebné znovu inicializovať generátory hodinového signálu a správne nastaviť regulátory napätia tak, aby mikrokontrolér mohol pokračovať v normálnej prevádzke. To, či je potrebná inicializácia hodinového signálu, záleží od konkrétneho mikrokontroléra. Posledným krokom je znovu inicializovať všetky periférie, ktoré knižnica odpojila pred prechodom do

režimu spánku. V prípade, že kontext niektorých periférií nebol zachovaný, tak sa tieto periférie musia taktiež inicializovať. Po inicializácii periférií môže mikrokontrolér pokračovať vo vykonávaní programu.

Proces prechodu do režimu nízkej spotreby, ktorý neuchováva kontext programu, je znázornený v diagrame D.1. Väčšinou sa režimy nízkej spotreby, ktoré nezachovávajú kontext, nazývajú Standby, ale niektorí výrobcovia ich pomenovali inak. Prechod do režimu standby sa od prechodu do režimu sleep líši hlavne v spôsobe prebudenia. Po prebudení z režimu standby sa mikrokontrolér reštartuje a program sa začína vykonávať od začiatku inicializácie. To znamená, že sa neuchoval kontext programu. Hneď po inicializácii je potrebné overiť, či bol systém prebudený zo spánku, alebo sa jedná o normálne zapnutie mikrokontroléra. V prípade, že sa systém prebudil z režimu standby, je nutné vymazať príznaky prebudenia a niektoré mikrokontroléry vyžadujú špecifické nastavenie konkrétnych periférií. Následne je zavolaná užívateľská funkcia, v ktorej si môže užívateľ obnoviť kontext uložený pred prechodom do režimu standby do nevolatilnej pamäte. Následne mikrokontrolér pokračuje v normálnom spracovávaní programu až do momentu, kedy užívateľ znovu požiada o prechod do režimu standby. Proces prechodu do režimu standby je rovnaký, ako do režimu sleep. Jediný rozdiel medzi nimi je, že knižnica zavolá užívateľskú funkciu, v ktorej si môže užívateľ uložiť dáta do nevolatilnej pamäte. Nasleduje prechod do režimu standby, v ktorom je väčšina modulov mikrokontroléra vypnutá. Po vygenerovaní prerušenia jednou z podporovaných periférií sa mikrokontrolér prebudí a spustí reštartovaciu sekvenciu rovnako, ako pri normálnom zapnutí.

3.5 Ukážka použitia rozhrania knižnice

Knižnicu je možné používať dvoma spôsobmi. Prvým spôsobom je používať iba ovládač hardvéru pre daný mikrokontrolér. Toto rozhranie je ideálne pre používateľa, ktorý chce mať úplnú kontrolu nad mikrokontrolérom. Nevyžaduje sa používanie vyšších vrstiev knižnice a riadenie spotreby energie je plne v réžii používateľa. Ukážky znázorňujú rozhranie ovládača pre mikrokontrolér STM32G4. Rozhranie ovládača sa mierne líši pre každé MCU, z dôvodu rozdielnych hardvérových implementácií režimov šetrenia energie.

V prvom rade musí prebehnúť inicializácia ovládača hardvéru, ktorej sa vloží konfiguračná štruktúra obsahujúca identifikačné číslo mikrokontroléra.

```
1 // Vytvorenie riadiacej štruktúry pre STM32G4.
2 struct LpmHwCtrl_s* stm32_ctrl = lpm_hw_ctrl_create(sLpmStm32g4Cfg);
3
4 // Inicializácia knižnice pre STM32G4.
5 _lpm_stm32g4_init(stm32_ctrl);
```

Pred vstupom do jednotlivých režimov nízkej spotreby je možné aktivovať časovač, ktorý systém prebudí po jeho vypršaní. Ak si užívateľ časovač nenastaví, systém sa musí prebudiť pomocou jedného z prerušení, ktoré môžu vygenerovať periférie.

```
1 // Nastavenie časovača na 1 sekundu
2 _lpm_stm32g4_set_wakeup_timer(1000)
```

Prechod do režimu spánku je možné zahájiť zavolaním funkcie z ovládača hardvéru. V parametri funkcie sa zvolí režim spánku, do ktorého má systém prejsť. Knižnica vykoná všetky potrebné operácie tak, aby mikrokontrolér úspešne prešiel do zvoleného režimu spánku. V tomto režime použitia má používateľ úplnú voľnosť nad tým, aký režim spánku si zvolí. Môže sa stať, že vyberie režim spánku, v ktorom nebudú aktívne programom vyžadované periférie. Ak tomuto chce používateľ predísť, môže využiť vyššie vrstvy knižnice, ktoré tieto závislosti kontrolujú a vyberú vhodný režim spánku.

```
// Prechod do režimu STOP0
_lpm_stm32g4_mode_enter(PM_STOP0);
```

Po prebudení knižnica automaticky obnoví napájacie domény a zdroje hodinového signálu tak, aby mikrokontrolér mohol ďalej pracovať.

3.5.1 Použitie abstraktného rozhrania

Abstraktné rozhranie úplne oddeľuje užívateľa od použitého hardvéru. Používateľ vôbec nemusí poznať, aké režimy spánku zvolený mikrokontrolér podporuje. Stačí definovať obmedzujúce podmienky a knižnica automaticky vyberie vhodný režim spánku, do ktorého mikrokontrolér prejde.

Po spustení mikrokontroléra je potrebné inicializovať vrstvu ovládača, ktorá inicializuje aj všetky nižšie vrstvy knižnice. Do inicializačnej funkcie je potrebné vložiť konfiguračnú štruktúru daného hardvéru, ktorá je súčasťou konfigurácie mikrokontroléra. Funkcia vráti ukazateľ na štruktúru ovládača. Tá sa následne využíva vo všetkých funkciách abstraktného rozhrania. Štruktúra obsahuje všetky dôležité informácie o aktuálnom režime spánku, v ktorom sa mikrokontrolér nachádza, a všetky konfiguračné parametre.

```
// Nájde konfiguráciu hardvéru podľa ID.
struct LpmHwCfg_s *hwCfg_p = lpm_hw_find(LPM_HW_ID_STM32G4);

// Vytvorenie ovládača pre STM32G4.
sLpmStm32g4Drv_p = lpm_hw_driver_create(hwCfg_p);
```

Ďalšou časťou inicializácie je registrácia všetkých periférií, ktoré aplikačná vrstva používa. Ku každej periférii musí užívateľ dodať vlastnú funkciu na inicializáciu a konfiguráciu periférie. Tieto funkcie sú knižnicou automaticky volané pri zmene režimu spánku.

```
// Registrácia periférií UART a I2C
lpm_hw_ctrl_resource_register(sLpmStm32g4Drv_p->ctrl, LPM_STM32G4_RSC_UART,
↪  _enable_uart2, _disable_uart2);

lpm_hw_ctrl_resource_register(sLpmStm32g4Drv_p->ctrl, LPM_STM32G4_RSC_I2C,
↪  _enable_i2c2, _disable_i2c2);
```

Po inicializácii môže používateľ začať používať knižnicu. V prípade, že chce aplikačná vrstva prejsť do režimu nízkej spotreby, musia sa najskôr vybrať periférie, ktoré môžu systém

prebudiť z režimu spánku. Tieto periférie ostanú, na rozdiel od ostatných, aktívne v režime spánku. Následne je možné zavolať funkciu na prechod do režimu nízkej spotreby energie. Jedným z parametrov funkcie je čas, po ktorom sa má mikrokontrolér zo spánku prebudiť v prípade, že ho žiadne vygenerované prerušenie neprebudilo skôr.

```
// Povolenie prebudenia pomocou periférie UART
lpm_hw_ctrl_resource_enable_wakeup(sLpmStm32g4Drv_p->ctrl, LPM_STM32G4_RSC_UART,
↪  true);

// Prechod do režimu spánku na maximálnu dobu 500 ms.
lpm_hw_driver_enter(sLpmStm32g4Drv_p, 500)
```

Používateľ môže pomocou udalostí zistiť, čo systém prebudilo z režimu spánku. Udalosti je možné získať rôznymi spôsobmi. Prvý spôsob je vyčítať udalosti pomocou obslužnej funkcie. Druhý spôsob je registrovať si notifikačnú funkciu, ktorá bude knižnicou zavolaná v momente, keď nastane udalosť.

```
if(lpm_hw_driver_read_event_mask(sLpmStm32g4Drv_p, LPM_HW_EVENT_UART_WAKEUP))
{
    printf("Systém bol prebudený prerušením z UART periférie");
}
```

Kapitola 4

Implementácia a testovanie

Obslužné funkcie pre riadenie jednotlivých režimov spánku som do knižnice implementoval pre tri rôzne mikrokontroléry, konkrétne STM32G4, NXP K60 a ESP32-S3. Tieto mikrokontroléry som vybral z dôvodu popularity, podobných podporovaných periférií a podobnej maximálnej frekvencie hodinového signálu. Podobnosť vybraných mikrokontrolérov je kľúčová na zabezpečenie porovnateľných výsledkov merania spotreby medzi nimi. Pre každý mikrokontrolér som naimplementoval obslužné funkcie, ktoré umožňujú prechod mikrokontroléra do všetkých režimov nízkej spotreby a prebudenie zo všetkých režimov. Knižnica taktiež vyžaduje implementáciu ovládania periférií časovačov, RTC a taktiež prebúdzanie pomocou vybraných periférií.

Knižnicu som implementoval pomocou šablóny LPM [13]. Túto šablónu som rozšíril o ďalšie funkcie a naimplementoval podporu pre jednotlivé mikrokontroléry. Zoznam zdrojových súborov, ktoré som implementoval v rámci diplomovej práce a súborov prevzatých zo šablóny LPM, je v prílohe F.

Testovanie implementácie spočíva v meraní spotreby mikrokontrolérov v jednotlivých režimoch spánku. Tým je možné overiť správnosť implementácie obslužných funkcií na prechod do jednotlivých režimov. Testovaním prebúdzania mikrokontrolérov pomocou jednotlivých periférií budem validovať proces prebudenia mikrokontrolérov. V druhej časti testovania implementujem rôzne testové úlohy, ktorých cieľom je otestovať implementáciu knižnice a porovnať spotrebu medzi mikrokontrolérmi v reálnych úlohách.

4.1 Implementácia podpory mikrokontrolérov do knižnice

Architektúra knižnice bola navrhnutá tak, aby ju bolo možné jednoducho rozšíriť o nové mikrokontroléry. Každý mikrokontrolér podporuje iné hardvérové funkcie a má inak navrhnutý systém riadenia spotreby. Z toho dôvodu je v knižnici navrhnuté jednotné rozhranie obslužných funkcií, ktoré poskytuje pomerne veľkú voľnosť v implementácii funkcií pre konkrétny mikrokontrolér. V prvom rade je potrebné definovať všetky režimy nízkej spotreby, ktoré mikrokontrolér podporuje. Taktiež je nutné definovať všetky periférie mikrokontroléra. Jeho identifikačné číslo je potrebné pridať do zoznamu identifikátorov v knižnici. Podľa tohto čísla sa bude ovládač mikrokontroléra identifikovať vo všetkých moduloch knižnice. Každý ovládač musí implementovať konfiguračnú štruktúru mikrokontroléra, súčasťou ktorej je tabuľka dostupných periférií vo všetkých režimoch nízkej spotreby MCU. Pomocou nich knižnica rozhoduje, do ktorého režimu nízkej spotreby môže mikrokontrolér prejsť na základe požiadaviek používateľa.

Okrem konfiguračnej štruktúry musí ovládač implementovať štyri hlavné obslužné funkcie hardvéru. Prvá je funkcia na inicializáciu, ktorá má za úlohu správne nastaviť všetky hardvérové prvky mikrokontroléra. Druhá je funkcia obsahujúca heuristiku, pomocou ktorej sa vyberá optimálny režim nízkej spotreby, do ktorého môže mikrokontrolér prejsť. Ďalšia je funkcia, pomocou ktorej mikrokontrolér prejde do požadovaného režimu spánku. Posledná obslužná funkcia má za úlohu spracúvať všetky prerušenia, ktoré boli vygenerované periférami.

```
// Inicializácia hardvérových prvkov mikrokontroléra.
int init(struct LpmHwCtrl_s* controller);

// Prechod do požadovaného režimu spánku.
int enter(struct LpmHwCtrl_s* controller, ePwrMode state);

// Výber optimálneho režimu nízkej spotreby.
ePwrMode get_optimal_state(struct LpmHwCtrl_s* controller);

// Spracovanie prerušenia vygenerovaných v mikrokontroléri.
int event_handler(int event_id);
```

4.1.1 STM32G4

Implementácia pre daný mikrokontrolér začína definovaním konfiguračných informácií. V prvom rade je potrebné vytvoriť v zozname identifikátorov MCU nový záznam.

```
typedef enum
{
    LPM_HW_ID_PROBE,
    LPM_HW_ID_NXP_K60,
    LPM_HW_ID_STM32G4,
    LPM_HW_ID_CNT
} lpmHwId_t;
```

Pomocou znalostí z referenčného manuálu je v konfiguračnej vrstve vytvorený zoznam všetkých režimov nízkej spotreby.

```
typedef enum ePwrMode {
    PM_RUN      = 0,  /**< Bežný režim (Run) */
    PM_SLEEP    = 1,  /**< Režim spánku (Sleep) */
    PM_STOP0    = 2,  /**< Stop 0 režim */
    PM_STOP1    = 3,  /**< Stop 1 režim */
    PM_STOP2    = 4,  /**< Stop 2 režim */
    PM_STANDBY  = 5,  /**< Režim Standby */
    PM_SHUTDOWN = 6   /**< Režim vypnutia (Shutdown) */
} ePwrMode_t;
```

Najťažšia časť implementácie konfiguračnej vrstvy je vytvorenie tabuľky 4.1, ktorá popisuje stav každej periférie v jednotlivých režimoch spánku. Pre každú periférium je táto informácia definovaná v referenčnom manuále. Bohužiaľ, nie vždy je úplne jasné, ktorá periféria bude v danom režime fungovať, často sú periférie závislé na napäťových doménach alebo obvodoch hodinového signálu. Takto vznikajú často viacnásobné závislosti, ktoré je zložité dekódovať a preniesť do tabuľky. Zatiaľ je tento proces plne manuálny, v budúcnosti by teoreticky bolo možné natrénovať neurónovú sieť, aby aspoň časť informácií vygenerovala automatizovane.

Periféria / Hodiny	Run	Sleep	Stop	Standby	Shutdown
GPIO	Active	Active	Sleep	Stopped	Stopped
USART	Active	Active	Active	Stopped	Stopped
LP UART	Active	Active	Active	Stopped	Stopped
I2C	Active	Active	Active	Stopped	Stopped
SPI	Active	Active	Sleep	Stopped	Stopped
CAN	Active	Active	Sleep	Stopped	Stopped
RTC	Active	Active	Active	Active	Active
HSI16	Active	Active	Stopped	Stopped	Stopped
HSE	Active	Active	Stopped	Stopped	Stopped
LSI	Active	Active	Active	Active	Stopped
LSE	Active	Sleep	Active	Active	Active

Tabulka 4.1: Prehľad stavu periférií a hodín v rôznych režimoch mikrokontroléra. (Tabuľka obsahuje iba najdôležitejšie periférie)

Tabuľku 4.1 je potrebné prepísať do konfigurácie v knižnici. Pre každú periférium v každom stave je možné definovať, či musí byť aktívna, nepovinne aktívna, vypnutá so zachovaním kontextu alebo vypnutá bez zachovania kontextu.

```
// Skrátená ukážka tabuľky s konfiguráciou periférií
static const _lpmStm32g4ModeTblCell_t
↪  _lpmStm32G4ModeTbl[LPM_STM32G4_RSC_CNT][LPM_STM32G4_MODE_CNT] = {
  /*                  RUN              SLEEP            STOP         STANDBY */
  /* GPIO     */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MSLP, _STM32G4_MOFF},
  /* USART    */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MON,  _STM32G4_MOFF},
  /* LP UART  */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MON,  _STM32G4_MOFF},
  /* I2C      */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MON,  _STM32G4_MOFF},
  /* SPI      */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MSLP, _STM32G4_MOFF},
  /* CAN      */ {_STM32G4_MON, _STM32G4_MON, _STM32G4_MSLP, _STM32G4_MOFF}
};
```

Režim sleep

V režime sleep je jadro mikrokontroléra pozastavené a čaká na prerušenie, ktoré ho prebudí. Pred vstupom do režimu spánku je potrebné zamaskovať všetky prerušenia, ktoré by mohli mikrokontrolér nechcene zobudiť z režimu spánku. Systick prerušenie je generované každú milisekundu, inkrementuje sa pomocou neho interné počítadlo milisekúnd od spustenia. Pred prechodom do režimu spánku sa musí vypnúť.

```
// Vypnutie systick prerušenia
CLEAR_BIT(SysTick->CTRL, SysTick_CTRL_TICKINT_Msk);
```

Pre zníženie spotreby knižnica automaticky vypne všetky periférie, ktoré si užívateľ zaregistroval a nepožaduje pomocou nich prebudenie systému. To sa robí automaticky pomocou funkcie, ktorá zavolá všetky užívateľom registrované metódy na ovládanie periférií.

```
_lpm_hw_ctrl_resource_disable(ctrl);
```

Následne je potrebné zvoliť, do ktorého z režimov sa má mikrokontrolér prepnúť. Na to slúži register SCB, ktorý riadi režimy spánku mikrokontroléra. Vymazaním príslušného bitu nastavíme cieľový režim sleep.

```
// Zakázanie hlbokého spánku (DEEPSLEEP bit)
CLEAR_BIT(SCB->SCR, SCB_SCR_SLEEPDEEP_Msk);
```

Posledným nutným krokom je zavolať inštrukciu WFI, ktorá pozastaví jadro mikrokontroléra. Jadro ostane pozastavené až do momentu, kedy bude vygenerované prerušenie systému. STM32 HAL knižnice poskytujú rozhranie v jazyku C nad týmito inštrukciami, takže nie je potrebné volať priamo assembler z C kódu.

```
__WFI();
```

Po prebudení mikrokontroléra je potrebné aktivovať systick prerušenie a inicializovať všetky periférie, ktoré sme vypli z dôvodu zníženia spotreby mikrokontroléra. Na to slúži pomocná funkcia v knižnici, ktorá zavolá užívateľmi registrované obslužné metódy.

```
// Povolenie prerušenia z SysTick časovača
SET_BIT(SysTick->CTRL, SysTick_CTRL_TICKINT_Msk);
// Zapnutie registrovaných periférií v knižnici
_lpm_hw_ctrl_resource_enable_disabled(ctrl);
```

Režimy STOP

Mikrokontrolér podporuje dva režimy STOP0 a STOP1. Hlavný rozdiel medzi nimi je nastavenie napäťového regulátora. Regulátor môže ostať v normálnom režime, alebo v režime nízkej spotreby. Podobne ako pri prechode do režimu sleep je potrebné deaktivovať Systick prerušenia. Pre režim STOP0 nie je potrebné nastavovať napäťový regulátor. Jeho konfigurácia ostáva rovnaká ako za behu, čo dovoľuje rýchlejšie prebudenie mikrokontroléra. V prípade prechodu do režimu STOP1 je potrebné prestaviť napäťový regulátor do režimu nízkej spotreby.

```
// Nastavenie napäťového regulátoru do režimu nízkej spotreby
MODIFY_REG(PWR->CR1, PWR_CR1_LPMS, PWR_CR1_LPMS_STOP1);
```

Následne je už proces rovnaký pre oba režimy STOP. V registry SCB sa výberom príslušného bitu vyberie režim hlbokého spánku pre jadro mikrokontroléra. Po nastavení stačí zavolať inštrukciu WFI a mikrokontrolér prejde do režimu STOP.

```
// Povolenie hlbokého spánku (DEEPSLEEP bit)
SET_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));
// Prechod do spánkového režimu
__WFI();
```

Po prebudení z režimu STOP je nutné obnoviť register SCB do pôvodného stavu, aby jadro mikrokontroléra neprešlo znovu do režimu hlbokého spánku.

```
// Zakázanie hlbokého spánku (DEEPSLEEP bit)
CLEAR_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));
```

V režimoch STOP sú úplne vypnuté domény hodinového signálu PLL, HSE a HSI. Po prebudení mikrokontroléra je potrebné znovu nakonfigurovať všetky domény hodinového signálu rovnako, ako po zapnutí mikrokontroléra. Užívateľ knižnice si môže registrovať vlastnú funkciu na konfiguráciu domén hodinového signálu. Ak ju neregistruje, tak sa nastavia na základné hodnoty. Následne sa nainicializujú všetky periférie, ktoré boli počas spánku vypnuté.

Režim standby

V režime standby sa v porovnaní s ostatnými režimami vypne väčšina periférií, jadro mikrokontroléra a neuchová sa kontext programu. Jediná pamäť, ktorá sa zachová, je malá časť SRAM2. Užívateľ si môže v knižnici registrovať funkciu, ktorá sa zavolá vždy predtým, než sa stratí kontext programu. V tejto funkcii si tak môže uložiť dôležité dáta do pamäte SRAM2. Sekvencia prechodu do režimu standby a následného prebudenia je znázornená v sekvenčnom diagrame v prílohe D.

Prvý krok prechodu do režimu standby je nakonfigurovať prípadne GPIO vstupy, ktoré môžu mikrokontrolér prebudiť. Iba vybraná skupina GPIO vstupov dokáže mikrokontrolér prebudiť z tohto režimu.

```
// Povolenie prebudenia z režimu standby pomocou GPIO vstupu
HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1_HIGH);
```

Pred prechodom do režimu standby je nutné vyčistiť bit indikujúci prebudenie z tohto režimu. Pokiaľ nebude vyčistený, tak sa mikrokontrolér nemusí správne prebudiť. Ak je aktívna požiadavka na prebudenie pomocou RTC modulu, musia byť aj jeho bity vynulované.

```
// Vymazanie wake-up flagu
__HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
// Vymazanie RTC wake-up flagu
RTC->ISR &= ~RTC_FLAG_WUTF;
```

Register SCB je nastavený rovnako, ako v režime STOP, aby jadro mikrokontroléra prešlo do hlbokého spánku. Nastavením bitu v registri PWR prestavíme napájanie do režimu standby. Zavolaním inštrukcie WFI prejde mikrokontrolér do režimu standby.

```
// Povolenie hlbokého spánku (DEEPSLEEP bit)
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
// Nastavenie STANDBY režimu
PWR->CR1 |= PWR_CR1_LPMS_STANDBY;
// Prechod do STANDBY režimu
__WFI();
```

Z režimu standby sa môže mikrokontrolér prebudiť iba pomocou RTC alarmu, alebo jedného zo skupiny podporovaných digitálnych vstupov. V tomto režime sa vypína napájanie pamäti RAM, takže sa neuchová kontext programu. Po prebudení je program spustený od začiatku, rovnako ako pri prvom zapnutí mikrokontroléra. Jediným rozdielom oproti normálnemu reštartovaniu je nastavený bit v registre PWR. Ten je potrebné vymazať po prebudení mikrokontroléru.

```
// Kontrola, či bol MCU prebudený z STANDBY režimu
if (__HAL_PWR_GET_FLAG(PWR_FLAG_SB) != RESET)
{
    // Vymazanie STANDBY flagu
    __HAL_PWR_CLEAR_FLAG(PWR_FLAG_SB);
}
```

Prebudenie pomocou časovača

Jednou zo základných funkcií našej knižnice je prebudenie mikrokontroléra po nastavenom čase. Knižnica udáva požiadavky na časovač, ktorý musí dosahovať rozlíšenie v milisekundách, aby si užívatelia mohli dostatočne presne nastaviť čas, za ktorý sa má mikrokontrolér zobudiť. Ideálnou voľbou je použiť časovač v RTC module. Ten je jediný, ktorý ostáva zapnutý v režime standby. Jeho vstupná frekvencia hodinového signálu je dostatočne rýchla na to, aby dosiahol rozlíšenie v jednotkách milisekúnd. Inak by sme museli používať normálny časovač vo všetkých ostatných režimoch spánku a RTC iba v režime standby. To je nutné na niektorých konkurenčných mikrokontroléroch, STM32G4 má v tomto výhodu.

Nastavenie alarmu RTC modulu je vďaka HAL ovládačom veľmi jednoduché. Stačí aktivovať prerušenie pri jeho vypršaní a nastaviť správnu frekvenciu hodinového signálu. Alarm je po vygenerovaní prerušenia potrebné deaktivovať, inak by generoval prerušenia periodicky.

```
void _lpm_stm32g4_set_wakeup_timer(int time_ms)
{
    // Nastavenie wakeup časovača s prerušením
    HAL_RTCEx_SetWakeUpTimer_IT(&hrtc, time_ms, RTC_WAKEUPCLOCK_RTCCLK_DIV16);
}
```

4.1.2 NXP K60

Postup implementácie pre mikrokontrolér NXP K60 je rovnaký, ako pre MCU STM32G4. V knižnici je potrebné definovať nový identifikátor, vytvoriť zoznam všetkých režimov nízkej spotreby energie, ktoré mikrokontrolér podporuje, a zoznam periférií. Taktiež je potrebné pomocou informácií z referenčného manuálu vytvoriť tabuľku 4.2 s podporovanými stavmi všetkých periférií podľa režimu spánku mikrokontroléra.

Periféria / Zdroj	WAIT	VLPW	VLPS	LLS
CPU	Active	Active	Sleep	Sleep
Bus Clock	Active	Active	Stopped	Stopped
RAM	Active	Active	Active	Active
Flash	Active	Stopped	Stopped	Stopped
GPIO	Active	Active	Active	Stopped
UART	Active	Active	Active	Stopped
I2C	Active	Active	Active	Stopped
SPI	Active	Active	Active	Stopped
RTC	Active	Active	Active	Active
Wakeup Logic	Active	Active	Sleep	Sleep

Tabulka 4.2: Prehľad stavu periférií a zdrojov v režimoch nízkej spotreby pre mikrokontrolér NXP K60.

Režim Wait a VLPW

Režim Wait je najjednoduchšie implementovateľný režim nízkej spotreby tohto mikrokontroléra. Jadro je v režime ľahkého spánku a zvyšné periférie ostávajú aktívne. Do tohto režimu mikrokontrolér prejde pomocou inštrukcie WFI.

```
__WFI();
```

Režim VLPW je podobný ako režim Wait, rozdielom je, že napäťové domény a zdroje hodinového signálu sú prepnuté do režimu veľmi nízkej spotreby. V tomto režime sa zníži napájacie napätie mikrokontroléra a zníži sa frekvencia na maximálne 4 MHz. Režim veľmi nízkej spotreby je potrebné povoliť v module SMC. Po povolení a nastavení správnej hodnoty v riadiacom registri PMCTRL prejde mikrokontrolér do režimu behu s veľmi nízkou spotrebou. Vložením inštrukcie WFI sa jadro mikrokontroléra prepne do režimu ľahkého spánku.

```
// Povolenie VLPR režimu (Very Low Power Run)
SMC->PMPROT |= SMC_PMPROT_AVLP_MASK;

// Prechod do VLPR režimu (RUNM = 10 - VLPR)
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(2);

// Čakanie na dokončenie prechodu do VLPR režimu
while ((SMC->PMSTAT & SMC_PMSTAT_PMSTAT_MASK) != 0x04) {
    // 0x04 = VLPR režim
}
```

Mikrokontrolér sa po prebudení nachádza v aktívnom režime s veľmi nízkou spotrebou. Nastavením registru PMCTRL prejde mikrokontrolér späť do aktívneho režimu a obnoví sa pôvodná frekvencia hodinového signálu.

```
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(0); // //
↪  Nastavenie režimu RUN (RUNM = 00 - Normal RUN)
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(0);

// Čakanie na potvrdenie návratu do RUN režimu
while ((SMC->PMSTAT & SMC_PMSTAT_PMSTAT_MASK) != 0x01) {
    // 0x01 = RUN režim
}
```

Režim STOP a VLPS

V režime STOP je jadro mikrokontroléra v režime hlbokého spánku a niektoré obvody hodinového signálu sú vypnuté. Mikrokontrolér je možné prebudiť vybranými perifériami.

Na prechod do režimu STOP je potrebné zapísať príslušné bity do registru PMCTRL. Tým sa deaktivujú nepotrebné hodinové signály. Zápis do registru SCB podobne, ako pri mikrokontroléri STM32, povolí režim hlbokého spánku jadra. Po spracovaní inštrukcie WFI sa jadro uspí.

```
// 2. Vybrať režim STOP v PMCTRL.STOPM (000 = STOP)
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | SMC_PMCTRL_STOPM(0x0); //
↪  STOPM = 000

// 3. Nastaviť SLEEPDEEP (potrebné pre všetky STOP režimy)
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

// 4. Čakať na prerušenie
__WFI();
```

Po prebudení mikrokontrolér automaticky prejde do režimu plného výkonu a všetky periférie si zachovajú svoj kontext, takže ich nie je potrebné znovu konfigurovať.

Režim VLPS je podobný, ako režim STOP. Jadro mikrokontroléra je v režime hlbokého spánku. Väčšina periférií je vypnutá a napäťové regulátory sú v režime nízkej spotreby. Prechod do režimu VLPS prebieha podobne, ako do režimu STOP. Jediným rozdielom je nastavenie bitov v registri PMCTRL jednotky riadenia spotreby. To zabezpečí, že sa prejde do správneho režimu. Jadro mikrokontroléra prejde do režimu hlbokého spánku nastavením hodnoty v registri SCB. Pomocou inštrukcie WFI prejde systém do režimu VLPS.

```
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | SMC_PMCTRL_STOPM(0x2); // STOPM = 010

// 2. Nastaviť SLEEPDEEP bit v System Control Register (SCR), aby sa umožnil STOP mód
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

// 3. Čakať na prerušenie - CPU teraz vstúpi do VLPS režimu
__WFI();
```

Po prebudení systém pokračuje v režime maximálneho výkonu. Všetky periférie si zachovali svoj kontext, takže ich nie je potrebné znovu nakonfigurovať.

Režim LLS

Režim LLS vypne väčšinu zdrojov hodinových signálov pre jadro a aj pre periférie. Systém už nemôže byť prebudený normálnymi prerušeniami pomocou jednotky NVIC. Mikrokontroléry NXP obsahujú jednotku LLWU, ktorá dokáže systém prebudiť pomocou vybraných periférií a nepotrebuje aktívny hodinový signál. Jadro systému je v režime hlbokého spánku. V tomto stave sa uchováva celá pamäť RAM a systém po prebudení pokračuje ďalšou inštrukciou v poradí, čiže nestráca kontext. Systém je možné prebudiť pomocou vybraných digitálnych vstupov, ktoré sú pripojené k jednotke LLWU, časovačom Low Power Timer (LPTMR) a modulom RTC.

Prechod do režimu LLS je podobný, ako do režimu VLPS. Hlavným rozdielom je, že stav LLS sa musí explicitne povoliť v jednotke riadenia spotreby. Nasleduje zápis do registru riadenia spotreby PMCTRL, nastavenie typu režimu LLS a povolenie hlbokého spánku pre jadro ARM. Mikrokontrolér prejde do režimu LLS po zavolaní inštrukcie WFI.

```
// 1. Povoliť všetky módy vrátane LLS nastavením ochrany v registri SMC_PMPROT
SMC->PMPROT |= SMC_PMPROT_ALLS_MASK; // Allow LLS mode
// 2. Vybrať režim LLS v PMCTRL.STOPM (0b011 = LLS)
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | SMC_PMCTRL_STOPM(0x3); // STOPM = 011
// 3. Nastaviť režim LLS v STOPCTRL register
SMC->STOPCTRL = 0x00; // LLS bez sub-mode (LLSx = 0)
// 4. Nastaviť SLEEPDEEP bit, aby Cortex-M4 vedel, že ide o STOP mód
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
// 5. Čakať na prerušenie - prechod do LLS
__WFI();
```

Po prebudení je zachovaný kontext programu, celý obsah pamäte SRAM a taktiež konfigurácia periférií, čiže nie je potrebné nanovo inicializovať obsah ich registrov.

Režim VLLS

Mikrokontrolér podporuje režimy VLLS0 až VLLS3. Hlavným rozdielom medzi nimi je, že sa v režimoch VLLS3 a VLLS2 zachová obsah pamäte SRAM. Väčšina periférií mikrokontroléra je vypnutá, po prebudení je obsah ich registrov vynulovaný. Jadro Armv7E je v režime hlbokého spánku. Prebudenie je možné iba pomocou jednotky LLWU, podobne, ako v režimoch LLS. Po reštarte mikrokontrolér začína vykonávať reštartovaciu sekvenciu, čo znamená, že sa program začína vykonávať od začiatku. Sekvencia prechodu do režimu VLLS a následné prebudenie je znázornená v sekvenčnom diagrame v prílohe D.

Prechodu do režimu VLLS je podobný, ako do režimu LLS. Najskôr je potrebné povoliť v jednotke riadenia spotreby SMC prechod do režimov VLLS. Do ovládacieho registru PMCTRL sa nastaví hodnota podľa požadovaného režimu. Následne je potrebné v špeciálnom registri VLLSCTRL nastaviť, ktorý z režimov VLLS sa má použiť. Jadro mikrokontroléra sa nastaví do režimu hlbokého spánku a zavolaním inštrukcie WFI sa spustí prechod do režimu VLLS.

```
// Povolenie vstupu do nízkoenergetických režimov VLLS, LLS a VLPS
SMC->PMPROT = SMC_PMPROT_AVLLS_MASK | SMC_PMPROT_ALLS_MASK |
↪  SMC_PMPROT_AVLP_MASK;
// Nastavenie režimu STOP s hodnotou 0b100 (VLLS režim)
SMC->PMCTRL = (SMC->PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | SMC_PMCTRL_STOPM(0b100);
// Konfigurácia VLLS2 režimu
SMC->VLLSCTRL = SMC_VLLSCTRL_VLLSM(0x2);
// Povolenie hlbokého spánku (DEEPSLEEP bit)
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
// Prechod do režimu VLLS2
__WFI();
```

Po prebudení mikrokontrolér začne program vykonávať od začiatku. Po inicializácii systému je nutné skontrolovať register SRS0. Pokiaľ je v ňom nastavený príslušný bit, znamená to, že bol systém prebudený z režimu VLLS. V tom prípade je nutné vynulovať register REGSC, ktorý znovu aktivuje napájacie domény pre všetky periférie a mikrokontrolér tak začne správne fungovať.

```
// Kontrola, či došlo k prebudeniu z VLLS režimu
if (RCM->SRS0 & RCM_SRS0_WAKEUP_MASK)
{
    // Potvrdené prebudenie z VLLS režimu
    if (PMC->REGSC & PMC_REGSC_ACKISO_MASK)
        // Zrušenie izolácie po prebudení
        PMC->REGSC |= PMC_REGSC_ACKISO_MASK;
}
```

Inicializácia jednotky LLWU

Mikrokontroléry NXP obsahujú jednotku LLWU, ktorá má za úlohu prebudiť systém z režimov hlbokého spánku LLS a VLLS. Jednotka podporuje prebudenie pomocou digitálnych vstupov, pre ktoré umožňuje nastaviť prebudenie pomocou nábežnej alebo ústupnej hrany signálu. Taktiež podporuje prebudenie systému pomocou časovačov LPTMR a RTC.

Jednotku LLWU je nutné pred použitím správne nakonfigurovať. Pokiaľ nie je nakonfigurovaná, mikrokontrolér sa nikdy neprebudí z režimov hlbokého spánku. Zápisom do registru SIM povolíme napájanie a zdroj hodinového signálu pre jednotku LLWU, ktorá ostane aktívna aj v režimoch hlbokého spánku. V registri ME je možné povoliť prebudenie pomocou podporovaných vstupov. V knižnici je podporované prebudenie pomocou RTC a digitálnych vstupov. Ako posledný krok je potrebné povoliť prerušenia generované jednotkou LLWU.

```
// Povolenie hodinového signálu pre LLWU modul
SIM->SCGC4 |= SIM_SCGC4_LLWU_MASK;
/* Konfigurácia RTC ako wakeup zdroja cez LLWU modul */
LLWU->ME |= LLWU_ME_WUME5_MASK;
// Povolenie prerušenia pre LLWU
NVIC_EnableIRQ(LLWU_IRQn);
```

Prebudenie pomocou časovača LPTMR

Časovač LPTMR poskytuje čas s rozlíšením menším, než jedna milisekunda. To spĺňa požiadavky, ktoré sú definované v našej knižnici. Časovač LPTMR funguje v režimoch Wait, STOP a VLPS. Keďže tento časovač nefunguje v režimoch LLS a VLLS, implementácia musí využívať oba časovače a automaticky vybrať ten správny, podľa požadovaného režimu spánku.

Konfigurácia časovača je jednoduchá, ovládače sú dostupné z HAL knižnice. Časovaču stačí nastaviť správne delič hodinového signálu, povoliť jednotlivé prerušenia a aktivovať ho.

```
// Povolenie prerušenia časovača LPTMR
LPTMR_EnableInterrupts(LPTMR0, kLPTMR_TimerInterruptEnable);
// Povolenie prerušenia pre LPTMR
EnableIRQ(LPTMR0_IRQn);
// Nastavenie časového obdobia časovača LPTMR
LPTMR_SetTimerPeriod(LPTMR0, time_ms);
// Spustenie časovača LPTMR
LPTMR_StartTimer(LPTMR0);
```

Po prebudení sa musí časovač vypnúť, inak by periodicky generoval prerušenia vždy po vypršaní nakonfigurovaného času.

Prebudenie pomocou časovača RTC

RTC môže ostať aktívny aj v režimoch LLS a VLLS. Je to jediná možnosť, ako v týchto režimoch prebudiť mikrokontrolér po uplynutí stanoveného času. Jeho hlavnou nevýhodou

je, že má rozlíšenie 1 sekundu, čo je nevýhoda oproti konkurenčným mikrokontrolérom. Tie majú rozlíšenie RTC časovača menej než 1 milisekundu. To znamená, že systém je možné po vstupe do režimov hlbokého spánku prebudiť časovačom najskôr za 1 sekundu. Toto obmedzenie je potrebné definovať do heuristickej funkcie, ktorá vyberie vhodný režim spánku podľa vstupných kritérií.

V inicializácii je v module RTC nutné zapnúť vstup hodinového signálu, vyčistiť registre indikujúce vygenerované prebudenie a aktivovať interný 32 KHz oscilátor, ktorý sa využíva na počítanie času. V prípade, že je čas v registri TSR neplatný, vynulujeme ho. Pre toto použitie nie je nutné v registri TSR udržiavať platný dátum a čas, stačí ho vynulovať, keďže chceme časovač nastavovať len v relatívnych časových prírastkoch. Zápisom do registru SR sa spustí počítadlo v module RTC.

```
// Povolenie hodinového signálu pre RTC modul
SIM->SCGC6 |= SIM_SCGC6_RTC_MASK;
// Zrušenie softvérového resetu RTC
RTC->CR &= ~RTC_CR_SWR_MASK;
RTC->CR = 0;
// Kontrola, či je 32kHz oscilátor vypnutý
if (!(RTC->CR & RTC_CR_OSCE_MASK))
{
    // Zapnutie 32kHz oscilátora
    RTC->CR |= RTC_CR_OSCE_MASK;
}

// Kontrola, či je čas neplatný (po resete)
if (RTC->SR & RTC_SR_TIF_MASK)
{
    // Nastavenie času na nulu, ak je neplatný
    RTC->TSR = 0;
}
// Povolenie časového počítadla RTC
RTC->SR |= RTC_SR_TCE_MASK;
```

Po inicializovaní stačí nastaviť požadovaný čas prebudenia a povoliť prerušenia pre RTC. Pokiaľ je systém v jednom z režimov hlbokého spánku a uplynie nastavený čas, jednotka LLWU systém prebudí.

```
// Inicializácia štruktúry pre konfiguráciu alarmu
rtc_datetime_t alarm_config;
RTC_GetDatetime(RTC, &alarm_config);

// Konfigurácia RTC alarmu na základe parametra sleep_time_seconds
alarm_config.second += sleep_time_seconds;
RTC_SetAlarm(RTC, &alarm_config);

// Povolenie alarmovej interrupcie
RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

// Povolenie prerušenia RTC
EnableIRQ(RTC_IRQn);
```

Systém prebudený z režimu VLLS začína vykonávať program od začiatku. Pomocou registru SRS0 je možné detekovať, že systém bol prebudený z režimu VLLS. Pokiaľ prebudenie spôsobí RTC alarm, tak je v registri RTC nastavený príslušný bit. Ten je potrebné vymazať, inak je RTC prerušenie cyklicky volané a systém sa nezvládne správne nainicializovať. Táto sekvencia sa musí vykonať ihneď po prebudení systému.

```
// Kontrola, či bolo MCU prebudené z wakeup zdroja
if (RCM->SRS0 & RCM_SRS0_WAKEUP_MASK)
{
    // Kontrola, či bol zdrojom prebudenia časovač
    if (RTC->SR & RTC_SR_TAF_MASK)
    {
        // Vynulovanie registra TAR
        RTC->TAR = 0;
    }
}
```

4.1.3 ESP32-S3

Implementácia pre mikrokontrolér ESP32-S3 je odlišná od zvyšku mikrokontrolérov, keďže jeho natívne ovládače ESP-IDF využívajú systém RTOS. Cieľom tejto implementácie je ukázať, že do knižnice na riadenie spotreby je možné implementovať aj mikrokontrolér, ktorý využíva operačný systém. Mikrokontrolér má len dva stavy nízkej spotreby energie. Implementácia režimov spánku do knižnice je jednoduchšia, ako v prípade mikrokontrolérov STM32 a NXP keďže ESP-IDF rieši väčšinu vecí automaticky.

Zostavenie tabuľky závislosti periférií na jednotlivých režimoch spánku je v prípade ESP32-S3 jednoduché, väčšina periférií dokáže systém prebudiť len v režime ľahkého spánku. Prehľad najdôležitejších periférií je zobrazený v tabuľke 4.3. V hlbokom spánku môže mikrokontrolér prebudiť len periféria RTC a koprocesor s nízkou spotrebou energie.

Periféria / Hodiny	Light Sleep	Deep Sleep
GPIO	Active	Active
UART	Optional	Stopped
SPI	Optional	Stopped
I2C	Optional	Stopped
Wi-Fi	Optional	Stopped
Bluetooth	Optional	Stopped
RTC	Active	Active
Timers	Optional	Stopped
ULP Co-proc.	Optional	Active
Flash	Active	Sleep

Tabulka 4.3: ESP32-S3: Stav periférií v režimoch Light Sleep a Deep Sleep.

Režim light sleep

V režime light sleep je jadro mikrokontroléra v režime ľahkého spánku. Zdroje hodinového signálu a periférie môžu byť aktívne, závisí to hlavne od konfigurácie. Prechod do tohto režimu je vďaka obslužným funkciám ESP-IDF veľmi jednoduchý.

```
// Prechod do režimu ľahkého spánku.
esp_light_sleep_start();
```

Pred prechodom do spánku je možné povoliť, ktoré periférie môžu mikrokontrolér prebudiť pomocou obslužných funkcií. Tieto periférie a ich zdroje hodinového signálu zostanú v prípade povolenia aktívne. ESP-IDF taktiež podporuje jednoduchú konfiguráciu RTC časovača, ktorý systém zobudí po uplynutí nastaveného času.

```
// Nastavenie časovača na 5 sekúnd
esp_sleep_enable_timer_wakeup(5 * 1000000);

// Nastavenie prebudenia pomocou GPIO vstupu
esp_sleep_enable_gpio_wakeup();
```

Po prebudení je zachovaný kontext programu a všetkých periférií. Mikrokontrolér automaticky pokračuje vo vykonávaní programu bez nutnej konfigurácie.

Režim deep sleep

V režime hlbokého spánku je jadro mikrokontroléra úplne vypnuté, tak ako väčšina zdrojov hodinového signálu. Väčšina periférií systému je odpojená od napájania. Prebudenie je podporované len pomocou RTC a koprocesora. Pamäť RAM je taktiež odpojená od napájania, takže sa kontext systému nezachová.

Pred prechodom do režimu hlbokého spánku stačí nastaviť časovač na požadovaný čas prebudenia, prípadne aktivovať koprocesor. Prechod do režimu hlbokého spánku je jednoduchý, keďže ESP-IDF vykoná všetky potrebné úkony automaticky.

```
// Prechod do režimu Deep sleep
esp_deep_sleep_start();
```

Po prebudení mikrokontrolér začína vykonávať program od začiatku, čiže sa stratil celý kontext programu a všetkých periférií. ESP-IDF ponúka obslužnú funkciu, pomocou ktorej je možné zistiť, či sa jedná o normálny štart mikrokontroléra alebo prebudenie pomocou jednej z podporovaných periférií.

```
// Vyčítanie zdroja prebudenia
esp_sleep_wakeup_cause_t cause = esp_sleep_get_wakeup_cause();
switch (cause) {
    case ESP_SLEEP_WAKEUP_TIMER:
        printf("Prebudenie pomocou časovača\n");
        break;
}
```

ESP32-S3 ponúka pamäť RTC RAM, do ktorej si môže užívateľ zapísať dáta, ktoré sa zachovajú aj po prebudení z režimu hlbokého spánku. Prostredie ESP-IDF poskytuje makro, ktoré nastaví prekladač, aby premennú uložil do danej pamäte.

```
// Vytvorenie premennej v pamäti RTC RAM
RTC_DATA_ATTR static int boot_count = 0;
```

4.2 Testovanie knižnice

Validácia úspešného prechodu mikrokontroléra do režimu spánku je zložitá. Vo väčšine režimov prestane fungovať debugger, takže týmto spôsobom sa nedá stav systému overiť. Najlepšia je možnosť priamo merať spotrebu mikrokontroléra s požadovanou presnosťou. Na testovanie som využíval ampérmeter Nordic Power Profiler Kit II [9], ktorý dokáže merať prúd s presnosťou v nanoampéroch. Jeho maximálna frekvencia merania je 100 000 Hz, čo ho robí ideálnym na zaznamenanie rýchlych prechodov medzi jednotlivými stavmi. Vybrané mikrokontroléry som testoval na vývojových sadách, ktoré sú bežne dostupné v obchodoch. To bolo problematické, keďže vývojové sady obsahujú okrem mikrokontrolérov aj množstvo pomocných obvodov, ktoré spotrebúvajú energiu a typicky nie sú optimalizované na nízku spotrebu. Tento parazitný odber ovplyvňoval výsledky meraní a znemožňoval priame porovnania medzi jednotlivými mikrokontrolérmi. Ideálnym riešením je navrhnúť vlastné dosky plošného spoja, ktoré budú mať napájanie mikrokontroléra úplne oddelené od zvyšku obvodov. Vďaka tomu by bolo možné ampérmetrom merať priamo spotrebu mikrokontroléra. Vývojové sady som upravil tak, aby bolo možné mikrokontrolér napájať osobitne, a dosiahol som tým výsledky merania, ktoré je možné porovnávať medzi jednotlivými mikrokontrolérmi.

4.2.1 Validácia implementácie knižnice

Implementácia úsporných režimov na mikrokontroléroch je zložitá hlavne preto, že sa obtiažne overuje, či mikrokontrolér a všetky jeho periférie správne prešli do režimu spánku. Počas implementácie som každý stav mikrokontroléra otestoval dvoma spôsobmi. V prvom rade som po prechode do režimu spánku nameral priemernú spotrebu mikrokontroléra a porovnal ju s hodnotami, ktoré udáva výrobca. V prípade, že mikrokontrolér mal násobne vyššiu spotrebu, ako bola očakávaná hodnota, postupne som testoval rôzne implementácie, kým som problém nevyriešil. Druhý spôsob validácie bolo postupne testovať prebudenie systému pomocou všetkých periférií, ktoré by mali prebudenie podporovať. V prípade, že sa systém nepodarilo prebudiť, znamenalo to, že implementácia daného režimu bola chybná. Po prebudení som otestoval opätovne funkcionalitu systému. V prípade, že po prebudení nejaká z periférií nefungovala správne, väčšinou to znamenalo, že nebol povolený niektorý z hodinových signálov, alebo ich frekvencia nebola správne nastavená. Zistil som, že prejsť do režimu spánku je väčšinou pomerne jednoduché, ale správne mikrokontrolér prebudiť tak, aby sa zachovala plná funkcionalita systému, je zložitejšie.

Súčasťou validácie bolo aj meranie latencie prebudenia. To znamená, ako rýchlo sa mikrokontrolér dokáže prebudiť po vygenerovaní prerušenia. Obecne platí, že mikrokontroléru trvá dlhšie prebudenie z úspornejších režimov spánku, pretože sa musí znovu inicializovať viacero periférií. Na meranie latencie prebudenia som pripravil test, k mikrokontroléru som pripojil tlačidlo na digitálny vstup, nastavil som prebudenie pomocou daného vstupu a mikrokontrolér som nastavil do testovaného režimu spánku. Následne som ho prebudil pomocou stlačenia tlačidla. Mikrokontrolér hneď po prebudení nastavil svoj digitálny výstup do logickej 1. Na osciloskope som sledoval signál z tlačidla a výstupný signál z mikrokontroléra a zmeral čas medzi nimi. Tento čas sa dá považovať za latenciu od vygenerovania prerušenia až po systém plne prebudený a schopný vykonávať užívateľský program.

Výsledky merania odoberaného prúdu a latencie

Na meranie odberu prúdu mikrokontrolérov som použil ampérmeter Nordic Power Profiler [9]. Všetky mikrokontroléry boli napájané z rovnakého 3.3 V zdroja. Zapojenie jednotlivých mikrokontrolérov a ampérmetra je znázornené v prílohe B. Na mikrokontrolér som nahral program, ktorý postupne prechádza cez všetky režimy spánku, v každom zotrvá pár sekúnd, prebudí sa a prejde do ďalšieho. Pomocou ampérmetru som získaval priebeh prúdu v čase, ktorý som pre každý režim spriemeroval a zapísal do tabuľky.

Z výsledkov merania v tabuľke 4.4 som zistil, že úspornejšie režimy spánku majú nižší odber prúdu, čo korešponduje s udávanými hodnotami od výrobcov. Meranie slúžilo aj na validáciu implementácie knižnice, z nameraných dát je možné určiť, že v jednotlivých režimoch spánku dosahujú všetky mikrokontroléry nižší odber prúdu ako v predchádzajúcom stave. Namerané hodnoty odoberaného prúdu sú porovnateľné s hodnotami udávanými v dokumentácii jednotlivých čipov, z čoho vyplýva, že obslužné funkcie sú v knižnici správne implementované.

Mikrokontrolér STM32G4 dosahuje bezkonkurenčne najnižší odoberaný prúd v režimoch Stop, Standby a Shutdown. Latencia prebudenia je rádovo vyššia v stavoch, v ktorých sa musí mikrokontrolér po prebudení reštartovať. Z toho vyplýva, že voľba správneho režimu spánku je ovplyvnená aj požadovanou latenciou prebudenia. Od latencie je odvodený aj minimálny čas, koľko musí mikrokontrolér v konkrétnom režime spánku zostať, aby bola dosiahnutá úspora energie.

MCU	Režim	Latencia prebudenia (ms)	Napájací prúd (mA)	Výrobcom deklarovaný napájací prúd (mA)
STM32G4	RUN	-	27.3	28.3
	Sleep	0.02	9.65	6.29
	Stop	0.12	0.095	0.08
	Standby	41	0.0015	0.0005
	Shutdown	41	0.0006	0.0001
ESP32-S3	RUN	-	60.1	23.8
	Sleep	0.51	0.91	0.3
	Standby	373	0.0086	0.008
NXP K60	RUN	-	42.0	81.0
	Wait	0.02	23.2	40.0
	VLPW	0.02	23.2	0.9
	Stop	0.02	0.98	1.3
	VLPS	0.02	0.32	0.25
	LLS2	0.15	0.289	0.25
	LLS3	0.15	0.289	0.25
	VLLS3	827	0.285	0.005
	VLLS2	827	0.283	0.002
	VLLS1	827	0.28	-

Tabulka 4.4: Porovnanie režimov nízkej spotreby pre STM32G4, ESP32-S3 a NXP K60. Napájací prúd udávaný výrobcom je orientačný.

ESP32-S3 dosahuje najhoršiu latenciu prebudenia zo všetkých režimov spánku. Z toho vyplýva, že tento mikrokontrolér nie je vhodný na úlohy, v ktorých je vyžadované rýchle prebudenie a reakcia na vygenerované prerušenie. Bohužiaľ, ESP32-S3 poskytuje len dva režimy spánku, čo znamená, že si používateľ nemôže vybrať režim, ktorý by vyhovoval všetkým požiadavkám danej úlohy.

NXP K60 dosiahol výbornú latenciu prebudenia vo všetkých režimoch, v ktorých sa zachováva kontext programu. Odber prúdu v režimoch VLLS3 až VLLS1 bol vyšší, ako bola definovaná hodnota v dokumentácii výrobcu. Implementáciu týchto režimov som viackrát overoval, ale nepodarilo sa mi zistiť, prečo je odber prúdu vyšší. Je možné, že prúd odoberá niektorý z komponentov pripojených ku mikrokontroléru. Ideálne by bolo navrhnúť vlastné PCB, na ktorej by bol len mikrokontrolér a napájanie. Tým by sa vylúčilo ovplyvňovanie merania ostatnými komponentami.

4.2.2 Návrh testovacích sád

Testovacie sady by mali otestovať riadenie spotreby energie v úlohách z reálneho sveta. Cieľom by malo byť otestovať vytvorenú knižnicu na riadenie spotreby energie na praktic-

kých príkladoch simulujúcich reálne použitie. Ďalším cieľom je porovnať implementáciu na jednotlivých mikrokontroléroch a zistiť, ktorý dosahuje najnižšiu spotrebu pri plnení definovaných úloh. S testovacou sadou by malo byť možné analyzovať jednotlivé režimy spánku a z výsledkov interpretovať, kedy sa oplatí použiť konkrétny režim spánku.

V teoretickej časti som popisoval testy ULPMark 2.4.1, ktoré sa špecializujú na hodnotenie spotreby mikrokontrolérov. Testové sady bohužiaľ nie sú verejné a počas písania práce sa mi nepodarilo získať financie na zakúpenie licencie. Rozhodol som sa preto navrhnúť vlastné testovacie sady, ktoré budú inšpirované sadou ULPMark. Testovacia sada bude po zverejnení práce pod licenciou open-source, takže ju budú môcť používať ostatní vývojári. Testovacia sada by mala mať jednoduché rozhranie, čo umožní jednoduchú implementáciu na ďalšie mikrokontroléry. To umožní porovnať výsledky tejto práce s ostatnými užívateľmi, ktorí sa túto testovaciu sadu rozhodnú použiť.

Testy využívajú UART periférium na výpis stavu testov a informovanie o dobe trvania výpočtu. K ampérmetru je pripojený jeden digitálny vstup, ktorý mikrokontrolér nastaví na logickú jednotku na začiatku testu a vynuluje na konci. Vďaka tomu sa dá presne vyhodnotiť dĺžka trvania testu a vypočítať spotrebovaná energia. Počas testu sa využíva jeden časovač, ktorý meria dobu trvania výpočtu.

Štandardizované rozhranie testov určuje tri druhy spánku. Prvým je ľahký spánok, v ktorom musia byť splnené nasledujúce podmienky:

- Aktivované prebudenie pomocou prijatého UART signálu
- Aktivované prebudenie pomocou digitálneho vstupu
- Zachovanie kontextu programu po prebudení
- Aktívny digitálny výstup
- Prebudenie pomocou časovača

Všetky ostatné periférie mikrokontroléra môžu byť v tomto režime spánku deaktivované, ale musia sa znovu zapnúť po prebudení. Požiadavky na tieto režimy som zvolil tak, aby bolo možné pre rôzne typy mikrokontrolérov vybrať vždy najvýhodnejší režim, pri ktorom splní definované podmienky a dosiahne najnižšiu spotrebu.

Druhým režimom je hlboký spánok, pre ktorý sú definované nasledujúce podmienky:

- Zachovanie kontextu programu po prebudení
- Aktívny digitálny výstup
- Prebudenie pomocou časovača

Režim standby musí splniť tieto podmienky:

- Prebudenie pomocou časovača

Po prebudení z režimu standby nemusí byť zachovaný kontext mikrokontroléra a program sa môže začať vykonávať od začiatku.

Jednotlivé testy môžu voľne využívať všetky tri režimy spánku podľa potreby. Konkrétna implementácia na danom mikrokontroléri rozhodne, do akého režimu mikrokontrolér prejde. Cieľom bolo zaistiť maximálnu kompatibilitu pre všetky mikrokontroléry tak, aby mal každý z nich možnosť dosiahnuť režim spánku s najnižšou spotrebou a výsledky testov tak boli čo najrelevantnejšie.

Testovacia sekvencia 1

Testovacia sekvencia číslo jedna sa zameriava hlavne na jadro mikrokontroléra. Počas testu sa nevyužívajú žiadne periférie okrem UART. Test je určený na porovnanie výpočtového výkonu jednotlivých mikrokontrolérov a porovnanie spotreby pri rôznych frekvenciách jadra. Na začiatku sa nastaví digitálny výstup do logickej 1, aby bolo možné detekovať začiatok testu v nameraných dátach. Následne sa spustí časovač a spustí sa algoritmus výpočtu mandelbrotových kvartálov. Konkrétny algoritmus nie je dôležitý, cieľ je otestovať výpočtovú rýchlosť mikrokontrolérov a ich spotrebu počas výpočtu. Po ukončení sa na UART vykreslí vypočítaná množina, ukončí sa časovač a čas výpočtu sa vypíše. Ukážka výpisu z testu je v prílohe G. Následne prejde mikrokontrolér do režimu ľahkého spánku, v ktorom zotrvá niekoľko sekúnd tak, aby celý cyklus testu trval 5 sekúnd. To znamená, že mikrokontroléry, ktoré budú počítať výsledok dlhšie, budú kratšiu dobu v spánku a naopak.

Krok	Popis
1. `set_gpio(true)`	Zapne výstupný signál indikujúci začiatok merania spotreby
2. `start_timer()`	Spustí časovač na meranie trvania výpočtu
3. `mandelbrot_benchmark()`	Spustí výpočtovo náročný test — Mandelbrotova množina
4. `printf(...)`	Vypíše trvanie výpočtu na UART
5. `stop_timer()`	Zastaví časovač
6. `enter_sleep(5000)`	Uvedie zariadenie do režimu spánku na 5 sekúnd
7. `set_gpio(false)`	Vypne výstupný signál indikujúci koniec merania

Tabulka 4.5: Postupnosť úloh v testovacej sade 1. na meranie spotreby mikrokontroléra.

Cieľom tejto testovej úlohy je odmerať efektivitu jednotlivých mikrokontrolérov počas výpočetne náročných úloh. Z výsledkov by malo byť možné zistiť, či je lepšie zvýšiť frekvenciu mikrokontroléra, aby sa výpočet vykonal rýchlejšie a mikrokontrolér prešiel do režimu spánku na dlhšiu dobu, alebo je nižšia frekvencia a s ňou spojená nižšia spotreba výhodnejšia aj za cenu dlhšej doby výpočtu a kratšieho spánku.

Testovacia sekvencia 2

Druhá testová sada pracuje so senzorom teploty pripojeným cez I2C zbernicu. Cieľom tohto testu je simulovať reálnu situáciu, kedy mikrokontrolér získa dáta z reálneho senzora, vypočíta z nich výslednú veličinu, podľa ktorej riadi digitálny výstup. Táto sekvencia by mala simulovať úlohy podobné regulácii systému.

Na začiatku testu pošle mikrokontrolér po I2C zbernici príkaz na začiatok merania teploty. Kým čaká na nameranie, prejde do režimu ľahkého spánku. Po prebudení vyčíta teplotu z I2C zbernice a vypíše ju. Následne sa spustí cyklus, kedy mikrokontrolér s periódou 1 ms nastaví digitálny výstup, prečíta jeho hodnotu a prejde do režimu spánku. Tento cyklus sa opakuje 10 krát a trvá približne 10 ms. Po ukončení tohto cyklu prejde mikrokontrolér do režimu ľahkého spánku na 80 ms. Po zobudení vypočíta iteratívne fibonacciho postupnosť, výsledok vypíše a prejde do režimu hlbokého spánku na zvyšok trvania testu. Ukážka výpisu z UART periférie je v prílohe G. Znovu platí, že mikrokontrolér, ktorý vykoná celú sekvenciu rýchlejšie, môže prejsť do režimu hlbokého spánku na dlhšiu dobu.

Cieľom tejto testovej úlohy je otestovať rýchlosť prechodov medzi režimami ľahkého spánku, ktoré sa cyklicky opakujú. Mikrokontrolér, ktorý dokáže rýchlejšie prejsť do režimu spánku, získava výhodu. Po prebudení musí systém komunikovať po I2C zbernici, čo zna-

Krok	Popis operácie
1.	`set_gpio(true)` — zapnutie GPIO (signál na spustenie testovania)
2.	`i2c_start_measurement()` — spustenie merania teploty pomocou I2C
3.	`enter_sleep(20)` — krátky light sleep počas čakania na meranie
4.	`i2c_read_measurement()` — čítanie nameranej teploty zo senzora
5.	`printf(...)` — výpis nameranej teploty
6.	`gpio_cycle()` — 10× cyklus: GPIO 1 → sleep → vyčítaj GPIO → GPIO 0 → sleep
7.	`set_gpio(true)` — opätovné zapnutie GPIO
8.	`printf(...)` — oznámenie ukončenia cyklu GPIO
9.	`enter_sleep(80)` — spánok na 80 ms (light sleep)
10.	`start_timer()` — štart časovača pre meranie doby výpočtu
11.	`fib_iter()` — výpočet 10 000× Fibonacci čísla (iteratívne)
12.	`printf(...)` — výpis výsledku a času výpočtu
13.	`stop_timer()` — zastavenie časovača
14.	`enter_deep_sleep(2000)` — deep sleep na 2000 ms
15.	`set_gpio(false)` — vypnutie GPIO po prebudení signalizuje koniec testu

Tabulka 4.6: Postupnosť úloh v testovacej sade 2. na meranie spotreby mikrokontroléra.

mená, že musí počas spánku zostať aktívna, alebo sa musí znovu inicializovať, čo bude mať negatívny vplyv na spotrebu.

Testovacia sekvencia 3

Tretí test sa zameriava na otestovanie režimov spánku, v ktorých sa nezachováva kontext programu a mikrokontrolér sa po prebudení reštartuje. Testová úloha najskôr spustí meranie a vyčíta teplotu z externého senzora rovnako, ako v úlohe 2. Po vypísaní teploty prejde mikrokontrolér do režimu najhlbšieho spánku. Jediné, čo je vyžadované v tomto režime, je prebudenie pomocou časovača. Po prebudení sa mikrokontrolér reštartuje a test končí po inicializácii všetkých periférií.

Krok	Popis operácie
1.	`set_gpio(true)` — zapnutie GPIO pre signalizáciu/prúdové meranie
2.	`i2c_start_measurement()` — spustenie merania teplotného senzora
3.	`enter_sleep(15)` — krátky light sleep na čakanie počas merania
4.	`i2c_read_measurement()` — čítanie nameranej hodnoty z teplotného senzora
5.	`printf(...)` — výpis nameranej hodnoty v °C
6.	`enter_standby(10000)` — vstup do standby režimu na 10 sekúnd

Tabulka 4.7: Postupnosť úloh v testovacej sade 3. na meranie spotreby mikrokontroléra.

Cieľom tohto testu je otestovať režimy spánku, v ktorých sa neukladá kontext programu. Z dát chcem zistiť, v ktorých prípadoch sa oplatí mikrokontrolér do tohto režimu prepnúť, keďže po prebudení spotrebuje dosť energie na opätovnú inicializáciu celého programu.

4.2.3 Výsledky meraní

Všetky testy som postupne spustil na každom mikrokontroléri trikrát a výsledky som spriemeroval. Merania spotreby boli vykonané pomocou ampérmetru Nordic Power Profiler [9]. Všetky mikrokontroléry boli napájané z rovnakého 3.3 V zdroja. Zapojenie mikrokontrolérov, senzoru teploty a ampérmetra je znázornené v prílohe B. Namerané dáta som uložil do počítača a spracoval pomocou skriptu. Začiatok a koniec jedného behu testu bol indiko-

vaný pomocou digitálneho výstupu mikrokontroléra, ktorý bol pripojený k Power Profileru. Pomocou skriptu bola vyhodnotená celková spotrebovaná energia, doba trvania výpočtu a priemerný spotrebovaný prúd v jednotlivých režimoch. Každá testová úloha bola spustená na mikrokontroléry s tromi rôznymi frekvenciami hodinového signálu, aby bolo možné porovnať rozdiel spotreby a doby výpočtu.

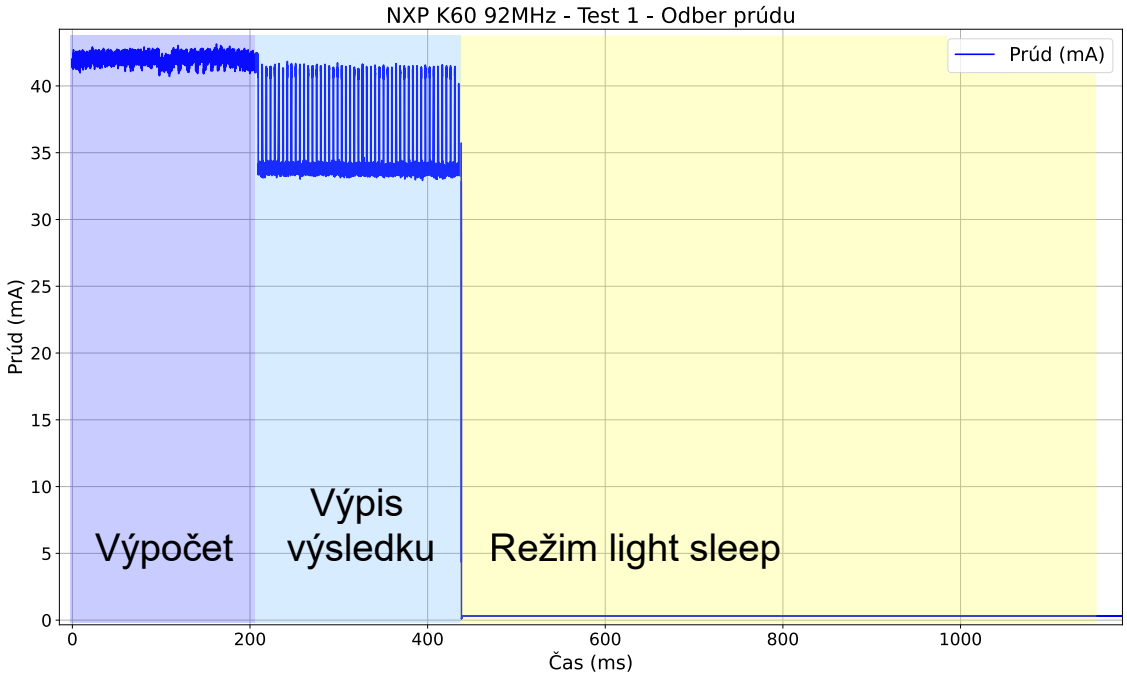
Výsledky testovacej sekvencie 1

V nameraných dátach v tabuľke 4.8 najskôr analyzujem rýchlosť výpočtu. Najkratšiu dobu výpočtu mal mikrokontrolér ESP32-S3. Zaujímavosťou je, že sa jeho doba výpočtu zmenila len minimálne so znižujúcou sa frekvenciou. Druhým najrýchlejším bol STM32G4 a najpomalší NXP K60, ktorého rýchlosť výpočtu bola najviac ovplyvnená znížením frekvencie. Celkovo najnižšiu spotrebovanú energiu dosiahol mikrokontrolér STM32G4 s frekvenciou 16 MHz. V grafe 4.1 je zobrazený priebeh odoberaného prúdu počas jednotlivých častí testu.

MCU	**Frekv. (MHz)**	**Energia (J)**	**Trvanie výpočtu (ms)**	**Prúd počas výpočtu (mA)**	**Prúd v light spánku (mA)**
STM32G4	164	0.2282	222	27.27	13.05
STM32G4	96	0.1149	228	14.51	6.52
STM32G4	16	0.0322	317	3.439	1.83
NXP K60	92	0.0604	437	42.04	0.313
NXP K60	62	0.0529	561	27.60	0.313
NXP K60	21	0.0486	1280	10.78	0.315
ESP32-S3	240	0.0589	215	60.07	0.980
ESP32-S3	160	0.0509	215	49.10	0.975
ESP32-S3	80	0.0410	220	35.75	0.962

Tabulka 4.8: Porovnanie spotreby mikrokontrolérov v testovacej sekvencii 1.

Mikrokontrolér STM32G4 dosiahol výrazne nižšiu spotrebu pri znížení frekvencie. Jemu, ako jedinému z testovaných mikrokontrolérov, klesal spotrebovaný prúd v režime ľahkého spánku pri znižovaní frekvencie. Z toho vyplýva, že mal aktívny hlavný zdroj hodinového signálu, ktorý bol využívaný niektorými periférami. Mikrokontrolér využíval režim Sleep, keďže ako jediný vyhovoval všetkým definovaným podmienkam. Pri zvyšovaní frekvencie sa rapídne zvyšoval odber prúdu počas výpočtu, ale doba výpočtu ostávala prakticky rovnaká. To znamená, že výpočet nebol obmedzený rýchlosťou aritmetického počítania, ale skôr pamäťou. Z toho vyplýva, že sa z hľadiska spotreby energie neoplatí zvyšovať frekvenciu, pokiaľ nie je spustený algoritmus optimalizovaný na danú platformu.



Obrázek 4.1: Meranie spotreby počas testovacej sekvencie 1. (Graf je priblížený, test ďalej pokračuje v režime light sleep)

NXP K60 dosiahlo najnižší odber prúdu v režime ľahkého spánku pri aktívnych všetkých periférіach. Bohužiaľ, doba výpočtu bola oproti konkurencii tak veľká, že ostatné mikrokontroléry dosiahli nižšiu celkovú spotrebu. Znižovanie frekvencie aj v tomto prípade prinieslo zlepšenie spotreby energie, aj keď sa doba výpočtu násobne predĺžila a mikrokontroléru tak ostal menej času v režime ľahkého spánku.

Znižovanie frekvencie mikrokontéru ESP32-S3 vôbec neovplyvňovalo dobu výpočtu, takže je jasné, že s nižšou frekvenciou dosiahol nižšiu spotrebu. V prípade tohto mikrokontroléru by bolo vhodné zvoliť algoritmus s dlhšou dobou výpočtu, keďže testová úloha bola vyriešená veľmi rýchlo. ESP32-S3 dosahovalo najmenší rozdiel v odbere prúdu za behu medzi jednotlivými frekvenciami. Ostatné mikrokontroléry dosahovali násobne nižší odber prúdu pri znížených frekvenciách.

Hlavný záver z meraní na prvej testovacej sade je, že všetky tri mikrokontroléry dosiahli nižšiu celkovú spotrebu, keď sa znížila frekvencia ich hodinového signálu. V prípade, že užívateľ nekladie nároky na dobu výpočtu, tak sa z hľadiska spotreby oplatí znížiť frekvenciu za cenu dlhšej doby výpočtu na všetkých troch platformách. Ďalším zistením je, že ESP32 a NXP K60 dosahujú podstatne nižší odber prúdu v režime ľahkého spánku s aktívnymi periférіami, ako STM32G4.

Výsledky testovacej sekvencie 2

V druhej testovej úlohe dosiahol najnižšiu spotrebovanú energiu mikrokontrolér STM32G4 s frekvenciou 16 MHz. ESP32-S3 a NXP K60 na tom boli veľmi podobne so spotrebovanou energiou, aj keď NXP K60 strávilo oveľa viac času výpočtom fibonacciho postupnosti. V tomto teste mal najväčšiu nevýhodu mikrokontrolér ESP32-S3, keďže podporuje len jeden režim spánku, v ktorom sa nestratí kontext aplikácie, takže pre neho testovaný režim light

sleep a deep sleep znamenal ten istý režim nízkej spotreby mikrokontroléra. Kvôli tomu mal vysoký odber prúdu v deep sleep, a teda aj vyššiu celkovú spotrebu. Ukážka priebehu odoberaného prúdu v 2. teste je v grafe 4.2.

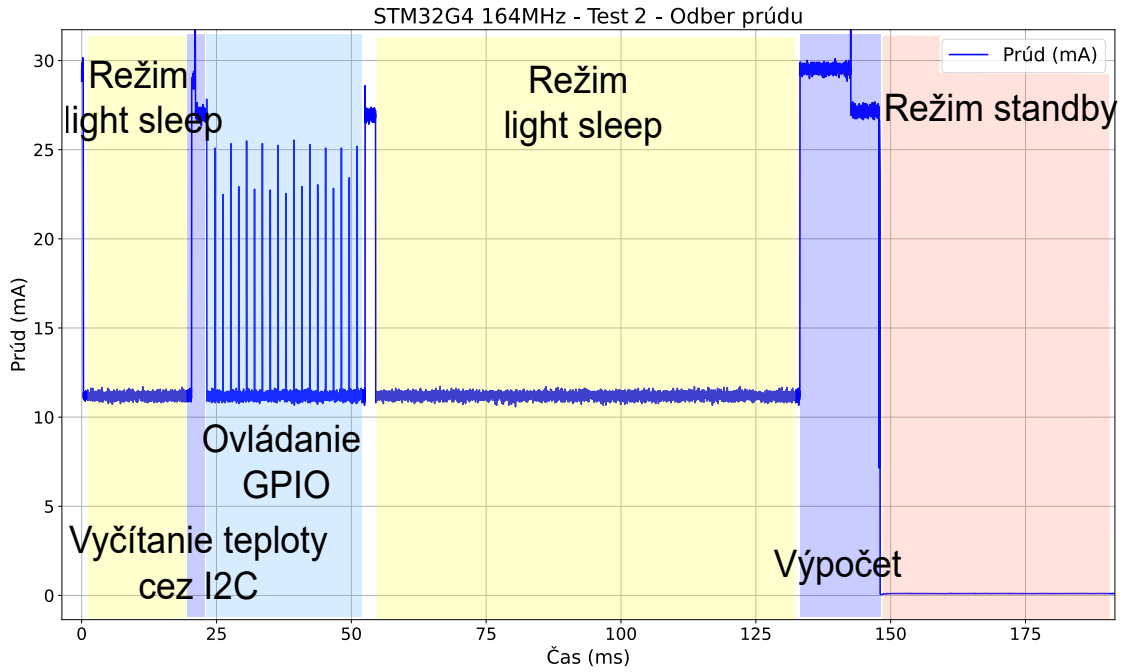
MCU	Frekv. (MHz)	Energia (J)	Trvanie výpočtu (ms)	Prúd počas výpočtu (mA)	Prúd v Light sleep (mA)	Prúd v Deep sleep (mA)
STM32G4	164	0.00728	14	27.27	11.1	0.099
STM32G4	96	0.00484	21	14.51	6.5	0.107
STM32G4	16	0.00268	104	3.439	1.832	0.10035
NXPK60	92	0.0091	57	42.04	0.315	0.281
NXPK60	62	0.0091	85	27.6	0.315	0.281
NXPK60	21	0.0097	243	10.78	0.315	0.281
ESP32S3	240	0.0108	8	60.07	0.963	0.964
ESP32S3	160	0.0105	11	49.1	0.963	0.964
ESP32S3	80	0.0111	21	35.75	0.963	0.964

Tabulka 4.9: Porovnanie spotreby mikrokontrolérov v testovacej sekvencii 2.

Z výsledkov merania v tabuľke 4.2.3 som zistil, že mikrokontrolér STM32G4 mal v celom rozsahu frekvencií nižšiu celkovú spotrebu, ako všetky ostatné mikrokontroléry. To je spôsobené hlavne najnižším odberom prúdu v režime deep sleep a taktiež pomerne rýchlym výpočtom testovej úlohy. Spotreba energie sa znižovala so znižujúcou sa frekvenciou hlavne preto, že frekvencia ovplyvňuje odber prúdu počas výpočtu, aj keď je mikrokontrolér v režime light sleep.

NXP K60 spotreboval veľké množstvo energie počas výpočtu, ktorý trval násobne viac času ako konkurencii. Rozdiel medzi odoberaným prúdom v light sleep a deep sleep bol len minimálny. Hodnoty odoberaného prúdu v týchto režimoch boli nižšie oproti mikrokontroléru ESP32. Spotrebovaná energia nebola závislá na frekvencii hodinového signálu, keďže v testovej úlohe nebolo veľa času stráveného počítaním. Väčšinu času boli mikrokontroléry v režime spánku, čo má najväčší dopad na spotrebu.

ESP32-S3 dosiahlo najvyššiu spotrebu zo všetkých testovaných mikrokontrolérov. Je to spôsobené hlavne tým, že mikrokontrolér podporuje len jeden režim spánku s uložením kontextu mikrokontroléra. Zatiaľ čo ostatné mikrokontroléry boli v režime deep sleep, ESP32-S3 bol stále v režime Light sleep. Odber energie nebol skoro vôbec závislý od frekvencie, čo je spôsobené podobne, ako v prípade NXP K60, tým, že sa mikrokontrolér nachádzal veľmi krátko v režime behu a väčšinu času strávil v režime spánku.



Obrázek 4.2: Meranie spotreby počas testovacej sekvencie 2. (Graf je priblížený, test ďalej pokračuje v režime deep sleep)

Záver z testu číslo 2 je, že pre úlohy, kde sú mikrokontroléry väčšinu času v režime spánku, je odoberaný prúd počas výpočtu mikrokontroléra zanedbateľný z hľadiska celkovej spotreby. V tom režime dosahoval mikrokontrolér STM32G4 bezkonkurenčne najnižšiu spotrebu. Ako jediný bol jeho výsledok ovplyvnený zmenou frekvencie, keďže od nej závisí aj spotreba v režime light sleep. Ostatné mikrokontroléry mali konštantnú spotrebu v režime light sleep, takže ich výsledná spotreba nebola závislá od frekvencie.

Výsledky testovacej sekvencie 3

V testovacej sade tri sú najväčšie rozdiely v spotrebe medzi jednotlivými mikrokontrolérmi. Namerané dáta z testov sú zobrazené v tabuľke 4.10. Najnižšiu spotrebu dosiahol STM32G4, hlavne vďaka nízkemu odberu prúdu v režime standby. V tomto režime sa neuloží kontext programu a systém sa musí po prebudení reštartovať. Preto sa spotreba merala až do momentu, kedy bol mikrokontrolér znovu pripravený vykonávať program. V tomto prípade závisí celková spotrebovaná energia aj od doby trvania inicializácie, ktorá sa taktiež meria počas testu. Priebeh odoberaného prúdu počas jednotlivých testov je zobrazený v grafe 4.3.

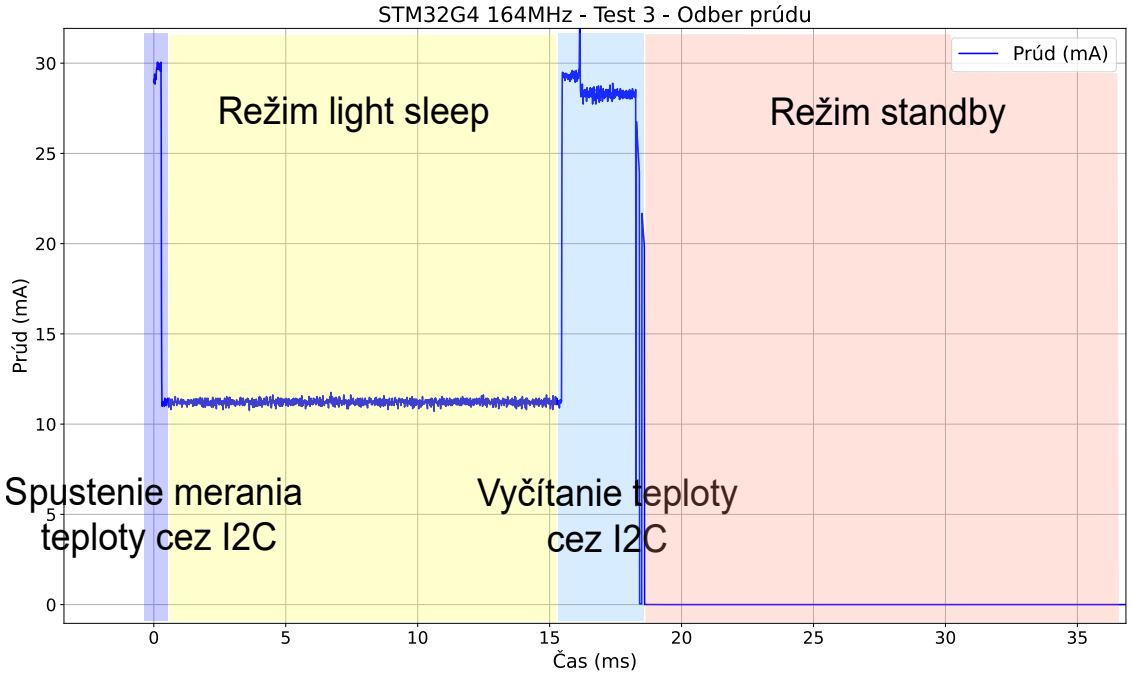
Z nameraných dát v tabuľke 4.10 je vidieť, že mikrokontrolér STM32G4 dosiahol najnižšiu spotrebu zo všetkých testovaných mikrokontrolérov. Taktiež mal najnižší odber prúdu v režime standby. Jeho inicializácia trvala oproti konkurencii najmenej času, čo malo tiež pozitívny vplyv na spotrebu. So znižujúcou sa frekvenciou klesala spotreba mikrokontroléra. To je spôsobené hlavne tým, že doba inicializácie bola pri rozdielnych frekvenciách konštantná a mikrokontrolér mal pri nižšej frekvencii nižší odber prúdu.

NXP K60 dosiahlo najvyššiu spotrebu z testovanej skupiny mikrokontrolérov. Jeho odber prúdu v režime standby bol rádovo vyšší oproti konkurencii. Inicializácia tohto mikrokontroléra trvala najviac času. V tomto prípade by sa viac oplatilo ostať v režime deep

MCU	Frekv. (MHz)	Energia (J)	Trvanie inicializácie (ms)	Prúd v light sleep (mA)	Prúd v standby (mA)
STM32G4	164	0.0047	40	11.22	0.00102
STM32G4	96	0.00266	41	6.52	0.001027
STM32G4	16	0.000746	49	1.834	0.001024
NXPK60	92	0.1102	827	0.318	0.28
NXPK60	62	0.1101	1240	0.3244	0.279
NXPK60	21	0.127	4120	0.32007	0.2802
ESP32S3	240	0.0774	373	1.595	0.009
ESP32S3	160	0.0678	384	1.595	0.009
ESP32S3	80	0.0568	386	1.595	0.009

Tabulka 4.10: Porovnanie spotreby mikrokontrolérov v testovacej sekvencii 3.

sleep, v ktorom má mikrokontrolér veľmi podobnú spotrebu, a po prebudení by nebolo nutné vykonať inicializáciu.



Obrázek 4.3: Meranie spotreby počas testovacej sekvencie 3. (Graf je priblížený, test ďalej pokračuje v režime standby)

Z meraní vyplýva, že mikrokontrolér STM32G4 dosahuje bezkonkurenčne najlepšiu spotrebu v režime standby a taktiež sa po prebudení z neho dokáže najrýchlejšie inicializovať. Na frekvencii mikrokontrolérov v tomto teste spotreba nezávisí, keďže sa nemení spotreba v režime standby, v ktorom sú mikrokontroléry najviac času.

4.2.4 Závery z meraní

Pomocou rozličných testových úloh a meraní som prišiel na nasledujúce závery, ktoré môže používateľ knižnice využiť pri implementácii svojej aplikačnej vrstvy:

- Pokiaľ mikrokontrolér dlhší čas počíta algoritmus, ktorý nie je dokonale optimalizovaný pre danú platformu, a doba výpočtu nie je hlavná priorita, tak sa oplatí znížiť frekvenciu mikrokontroléra, čo prinesie zníženie spotreby.
- Pokiaľ aplikácia vyžaduje dlhodobo zotrvať v režime light sleep s aktívnymi periférami, tak najnižšiu spotrebu dosahuje mikrokontrolér NXP K60.
- Ak nie je úloha výpočetne náročná, a mikrokontrolér môže na dlhšiu dobu prejsť do režimu deep sleep, tak spotreba nezávisí od jeho frekvencie.
- V prípade výpočetne nenáročnej úlohy sa najviac zníži spotreba prechodom do režimov deep sleep, pokiaľ ho mikrokontrolér podporuje.
- Pokiaľ aplikácia umožnuje prejsť do režimu standby, tak mikrokontrolér STM32G4 dosahuje bezkonkurenčne najnižšiu spotrebu.
- Do režimu standby sa oplatí prejsť, keď v ňom mikrokontrolér ostane aspoň pár sekúnd, keďže počas opätovnej inicializácie spotrebuje zanedbateľné množstvo energie.

Z výsledku testov som odmeral spotrebovanú energiu v jednotlivých režimoch spánku, analyzoval spotrebovanú energiu pri rôznych testových úlohách a rôznych frekvenciách, a zistil som, čo má najväčší vplyv na spotrebovanú energiu. Pre mikrokontroléry STM32G4 a ESP32-S3 by bolo vhodné upraviť testovú úlohu 1 tak, aby bola výpočtovo náročnejšia. Ostatné testy a merania dopadli podľa očakávaní. Z grafov jednotlivých meraní odberu prúdu je možné vyčítať, v akej časti testu sa mikrokontrolér nachádza. Jediná odľahlá hodnota v dátach, ktorá nesúhlasí s tvrdeniami výrobcu, je odber prúdu mikrokontroléru NXP K60 v režime standby. Toto meranie a implementáciu som viackrát kontroloval, ale odber prúdu bol vo všetkých prípadoch rovnaký.

4.3 Výsledky testovania

Pomocou testov som overil, že knižnica funguje správne, a je pomocou nej možné znížiť spotrebu mikrokontrolérov. V prvej časti testov som overil spotrebu v jednotlivých režimoch spánku. Výsledky statických testov ukázali, že sú jednotlivé režimy spánku v knižnici správne naimplementované a znižujú spotrebu energie mikrokontroléra. V tejto časti som overil, že sa mikrokontroléry dokážu prebudiť z jednotlivých režimov spánku a môžu pokračovať vo vykonávaní programu. Implementácia obslužných funkcií podporuje nastavenie časovačov a RTC modulu tak, aby sa mikrokontroléry dokázali prebudiť zo všetkých režimov po uplynutí požadovaného času. V rámci statických testov som taktiež nameral latenciu prebudenia mikrokontrolérov z jednotlivých režimov spánku. Tieto dáta môžu v budúcnosti slúžiť na vývoj heuristiky určenej na výber optimálneho režimu spánku na základe používateľských požiadaviek.

V druhej časti testovania som knižnicu validoval na úlohách simulujúcich dynamické prechody medzi režimami spánku. Z nameraných výsledkov je vidieť, že vyvinutá knižnica znížila spotrebu energie vo všetkých testových úlohách. Súčasťou tohto testovania je porovnanie jednotlivých mikrokontrolérov a identifikácia ich silných a slabých stránok z hľadiska

znižovania spotreby. Pomocou nameraných dát som vytvoril zoznam zásad, ktoré pomôžu vývojárovi systému dosiahnuť minimálnu spotrebu energie vzhľadom k typu úlohy, ktorú systém vykonáva.

Z testovania knižnice hodnotím, že je jednoduchá na použitie, aplikačné rozhranie je prehľadné a jednoduché na pochopenie pre používateľa, ktorý nepozná režimy spánku na danom type mikrokontroléru. Výhodou knižnice je, že používa architektúru viacerých vrstiev, ktoré je možné používať osobitne. Skúsení užívatelia tak môžu používať iba najnižšie vrstvy a dosiahnu tak plnú kontrolu nad režimami mikrokontroléra. Menej skúsení užívatelia môžu využiť plne abstraktné rozhranie a nemusia dopodrobna poznať mikrokontrolér, s ktorým pracujú.

Jednou z nevýhod aktuálnej implementácie knižnice je, že pre pridanie podpory nového mikrokontroléra je nutné manuálne zostaviť tabuľku stavov periférií v závislosti od režimu prevádzky mikrokontroléra. Zostavenie tejto tabuľky je časovo náročné a vyžaduje podrobnú znalosť referenčného manuálu mikrokontroléra. V budúcnosti by táto činnosť mohla byť aspoň z časti automatizovaná pomocou AI. Druhou nevýhodou v porovnaní s knižnicou Zephyr je, že vyvinutá knižnica neobsahuje ovládače pre jednotlivé periférie. Tie si musí vývojár doplniť sám, a následne musí do knižnice registrovať funkcie na zapnutie a vypnutie týchto periférií, ktoré potom knižnica používa. Pre tento kompromis som sa rozhodol z dôvodu, že poskytuje používateľovi možnosť používať preferované ovládače periférií a nie je nútený použiť tie, ktoré sú naimplementované v našej knižnici. Vytvoriť ovládače pre každú periférium každého podporovaného mikrokontroléra je časovo náročné, aj keď by to prinieslo ďalšie možnosti na zníženie spotreby periférií. Vďaka modulárnemu návrhu je knižnicu možné v budúcnosti o ovládače periférií rozšíriť. Ďalšou nevýhodou oproti konkurencii je, že naša knižnica nepodporuje výber optimálneho režimu spánku podľa latencie prebudenia. V rámci diplomovej práce nebol priestor túto funkciu pridať, ale knižnicu je možné vďaka systému heuristiky výberu režimu spánku v budúcnosti o túto funkciu rozšíriť.

Celkovo považujem implementáciu podpory riadenia spotreby mikrokontrolérov do knižnice LPM [13] za úspešnú. Podarilo sa mi splniť ciele, ktoré som si stanovil, knižnica je ľahko rozšíriteľná, modulárna a jednoduchá na použitie. Pomocou vyvinutých testových úloh som dokázal, že knižnicu je možné použiť v úlohách z reálneho sveta a umožňuje znížiť spotrebu energie celého systému. Do knižnice som v rámci diplomovej práce pridal podporu pre tri rôzne mikrokontroléry, z čoho vyplýva, že knižnica dokáže pracovať s rôznymi typmi mikrokontrolérov od rôznych výrobcov.

4.4 Plány do budúcna

Do knižnice na riadenie spotreby energie mikrokontrolérov som naimplementoval podporu pre mikrokontroléry STM32G4, NXP K60 a ESP32-S3. Implementácie obsahujú prechody do podporovaných režimov spánku. V aktuálnom stave si musí užívateľ vybrať sám, do ktorého z režimov spánku má mikrokontrolér prejsť, alebo môže využiť automatický systém, ktorý rozhoduje pomocou definovaných kritérií. Knižnicu by bolo vhodné rozšíriť o možnosť definovať pre každý režim spánku minimálny čas, koľko sa musí zotrvať v režime spánku, aby mal prechod do daného režimu zmysel z hľadiska spotreby. Každý režim má definovanú latenciu prebudenia, a ak je požadovaná dĺžka spánku nižšia alebo rovnaká ako latencia prebudenia, tak nemá význam do režimu spánku prechádzať, keďže sa mikrokontrolér musí hneď začať prebúdzať. Do konfiguračnej vrstvy knižnice by sa tak mohli pridať namerané latencie prebudenia z jednotlivých meraní a výber vhodného režimu spánku by mohol pri rozhodovaní využiť aj tento údaj.

Ďalšou zaujímavou možnosťou je do konfiguračnej vrstvy pridať odoberaný prúd v konkrétnom režime spánku. Tento údaj v kombinácii s latenciou prebudenia a požadovanou dobou v režime spánku by mohol umožniť presnejšie vypočítať, aký režim zvoliť tak, aby sa minimalizovala spotreba. Pokiaľ užívateľ vyžaduje prechod do režimu spánku na krátku dobu, tak sa neoplatí vybrať režim spánku s dlhou latenciou prebudenia, keďže mikrokontrolér strávi viac času prebúdzaním, ako v režime spánku. Pomocou odoberaného prúdu by bolo možné vypočítať celkovú spotrebu v režime spánku aj počas prebúdzania a zistiť tak, či je prechod do režimu spánku výhodný z hľadiska spotreby.

Implementáciu ovládačov pre nové typy mikrokontrolérov by mohlo uľahčiť natrénovanie umelej inteligencie, ktorá by dokázala automaticky generovať tabuľky stavov periférií v jednotlivých režimoch spánku z referenčných manuálov. Tento proces je v súčasnom stave manuálny a zaberá veľké množstvo času z dôvodu, že v referenčných manuáloch často nie je jasne napísané, ktoré periférie budú v jednotlivých režimoch fungovať. Umelá inteligencia by mohla tento proces aspoň z časti automatizovať, ale je otázne, či budú výsledné vygenerované tabuľky obsahovať pravdivé informácie. Vývojár by tieto tabuľky mohol kontrolovať a opraviť chybné údaje.

Dlhodobou úlohou by malo byť postupné rozširovanie podporovaných mikrokontrolérov o nové typy. Najviac času zaberie implementácia prvého mikrokontroléra od výrobcu, ktorý v knižnici ešte nie je podporovaný. Ovládače pre ďalšie typy mikrokontrolérov od rovnakého výrobcu už sú menej časovo náročné na implementáciu a väčšinou ich zvládne aj menej skúsený vývojár, keďže sa väčšinou jedná iba o úpravu už vytvorených funkcií. Podpora viacerých mikrokontrolérov môže priniesť viac používateľov knižnice, ktorí môžu následne knižnicu rozširovať o nové funkcie alebo podporované mikrokontroléry.

V neposlednom rade je cieľom zlepšiť podporu knižnice pre operačné systémy pracujúce v reálnom čase. Pomocou plánovača v operačných systémoch je možné automaticky prechádzať do režimov spánku bez užívateľského zásahu vždy, keď je mikrokontrolér v stave nečinnosti. Táto funkcia by v kombinácii s automatickým výberom optimálneho režimu spánku z našej knižnice mohla poskytnúť automatické zníženie spotreby systému bez nutného zásahu do vytvoreného systému.

Kapitola 5

Záver

V tejto diplomovej práci som sa venoval problematike riadenia spotreby mikrokontrolérov, s cieľom navrhnúť a implementovať knižnicu, ktorá umožní vývojárom jednoducho znížiť spotrebu mikrokontrolérov v ich aplikáciách. Cieľom bolo vytvoriť univerzálne riešenie, ktoré umožňuje jednoducho pracovať s režimami spánku mikrokontrolérov bez nutnosti podrobne poznať ich implementáciu.

V úvodnej časti diplomovej práce som vysvetlil problematiku znižovania spotreby mikrokontrolérov, definoval aké techniky sa na znižovanie spotreby využívajú, a určil, ktoré aplikácie sú vhodné na riadenie spotreby mikrokontrolérov. Zameral som sa na tri mikrokontroléry od popredných výrobcov, detailne som analyzoval hardvérovú podporu riadenia spotreby energie, jednotlivé režimy spánku a porovnal ich medzi sebou.

Analyzoval som dve populárne knižnice na riadenie spotreby mikrokontrolérov, ukázal som, ako sa s nimi pracuje z užívateľského hľadiska, a taktiež ich architektúru. Z analýzy týchto knižníc vznikli požiadavky na vlastnú knižnicu riadenia spotreby energie. Súčasťou teoretickej časti je aj kapitola o problematike merania spotreby mikrokontrolérov a ukážka testových úloh, ktoré sa používajú na porovnanie mikrokontrolérov z hľadiska spotreby.

Vytvoril som knižnicu na riadenie spotreby energie, ktorá vychádza zo šablóny LPM [13] vyvíjanej na Fakulte Informatiky VUT. Do knižnice som implementoval podporu pre vybrané mikrokontroléry s funkciami na obsluhu riadenia spotreby. Implementácia umožňuje prechod do jednotlivých režimov spánku a obsluhu hardvérových periférií, ktorú je potrebné po prebudení vykonať. Meraním spotreby mikrokontrolérov som validoval správnosť implementácie knižnice. Z nameraných dát vyplýva, že knižnica znižuje spotrebu mikrokontrolérov a spĺňa definované požiadavky.

Navrhol som testovacie sekvencie, ktoré reprezentujú využitie mikrokontrolérov v reálnych aplikáciách. Každá testovacia sekvencia sa zameriava na iné časti riadenia spotreby. Testovacie sekvencie sú voľne dostupné pre ďalších vývojárov, aby mohli porovnať spotrebu svojich systémov s mikrokontrolérmi použitými v tejto práci. Pomocou testovacej sady som nameral spotrebu energie pre vybrané mikrokontroléry. Z výsledkov vyplýva, že knižnica znižuje spotrebu vybraných mikrokontrolérov vo všetkých testových úlohách. Knižnica spolu s nameranými dátami tvoria základ, ktorý je možné ďalej rozširovať a vyvíjať.

Z výsledkov práce hodnotím, že som splnil ciele zadania, vyvinul som knižnicu na riadenie spotreby mikrokontrolérov, ktorá je pre používateľov jednoduchá na použitie, modulárna a ľahko rozšíriteľná pre ďalšie mikrokontroléry. V práci som vyhodnotil výhody a nevýhody navrhnutého riešenia a vytvoril zoznam možných vylepšení.

Literatura

[1] ARM Holdings. *Cortex-M4 Processor Technical Reference Manual* online. R0p1, r0p1. Cambridge, United Kingdom, březen 2010. Dostupné z: https://developer.arm.com/documentation/100166/latest/. [cit. 2025-01-10].

[2] EEMBC. The New World of Power Management: Why It Matters. *Elektroniknet* online. 1. El Dorado Hills, CA: EEMBC, Červenec 2015. Dostupné z: https://www.eembc.org/techlit/articles/elektroniknet_article_2015_07_21/index.html. [cit. 2024-12-11].

[3] EEMBC. *ULPMark: A Benchmark for Ultra-Low-Power Microcontrollers* online. 1. El Dorado Hills, CA: EEMBC, leden 2023. Dostupné z: https://www.eembc.org/ulpmark/. [cit. 2025-01-05].

[4] Liu, Z.; Shi, J. a Peng, W. Design of a Low-Power Humidity and Temperature Collector Based on STM32. *Proceedings of the Second International Conference on Mechatronics and Automatic Control* online. Cham: Springer International Publishing, Březen 2015. Dostupné z: https://doi.org/10.1007/978-3-319-13707-0_88. [cit. 2025-05-15].

[5] Munir, A.; Ranka, S. a Gordon Ross, A. High-Performance Energy-Efficient Multicore Embedded Computing. *IEEE Transactions on Parallel and Distributed Systems* online. Piscataway, NJ: IEEE, Duben 2012, sv. 23, č. 4. ISSN 1045-9219. Dostupné z: https://doi.org/10.1109/TPDS.2011.214. [cit. 2025-01-10].

[6] NXP Semiconductors. *Kinetis K60 Sub-Family Reference Manual* online. Rev. 6.1, 6.1. Eindhoven, Netherlands, srpen 2012. Dostupné z: https://www.nxp.com/products/K60_120. [cit. 2025-01-18].

[7] NXP Semiconductors. *Power Management for Kinetis MCUs* online. Rev. 2, 2.0. Eindhoven, Netherlands, duben 2015. Dostupné z: https://www.nxp.com/docs/en/application-note/AN4503.pdf. [cit. 2025-01-27].

[8] NXP Semiconductors. *AN13956: Overview of the NXP Kinetis MCU Series* online. 2.0. Eindhoven, Netherlands, květen 2020. Dostupné z: https://www.nxp.com/docs/en/application-note/AN13956.pdf. [cit. 2024-12-08].

[9] Semiconductor, N. *Power Profiler Kit II User Guide: Technical Documentation* online. 1. vyd., 1.0. Trondheim, Norway, květen 2025. Dostupné z: https://docs.nordicsemi.com/bundle/ug_ppk2/page/UG/ppk/PPK_user_guide_Intro.html. [cit. 2025-05-15].

[10] Shankari, K.; Fürst, J.; Wang, Y.; Bonnet, P.; Culler, D. et al. Zephyr: Simple, Ready-to-use Software-based Power Evaluation for Background Sensing Smartphone Applications. online, Prosinec 2018. Dostupné z: https://doi.org/10.13140/RG.2.2.36772.73605. [cit. 2024-10-27].

[11] STMicroelectronics. *STM32G4 System Power Control* online. 1.0. Geneva, Switzerland, březen 2020. Dostupné z: https://www.st.com/resource/en/product_training/STM32G4-System-Power_control_PWR.pdf. [cit. 2024-12-08].

[12] STMicroelectronics. *STM32G4 Series Advanced Arm®-based 32-bit MCUs Reference Manual* online. Rev. 8, RM0440. Geneva, Switzerland, únor 2024. 1–2138 s. Dostupné z: https://www.st.com/resource/en/reference_manual/rm0440-stm32g4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf. [cit. 2024-11-21].

[13] Strnadel, J. *LPM: Library for Power Management: Software Layer for Power Management of Embedded Systems* online. 1st Edition, 1.0. Brno, Czech Republic, květen 2025, revidováno 1. 5. 2025. Dostupné z: https://github.com/josef-strnadel/lpm. [cit. 2025-05-09].

[14] Systems, E. *ESP32-S3 Sleep Modes: Low Power Management for ESP32-S3* online. 1st Edition, 1.0. Edited by Espressif Systems. Shanghai, China, červen 2024, revidováno 1. 6. 2024. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/system/sleep_modes.html. [cit. 2025-05-09].

[15] Systems, E. *ESP32-S3 Technical Reference Manual: Technical Documentation for ESP32-S3* online. 1st Edition, 1.0. Edited by Espressif Systems. Shanghai, China, červen 2024, revidováno 1. 6. 2024. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf. [cit. 2025-05-09].

[16] Xhonneux, M.; Louveaux, J. a Bol, D. A Sub-mW Cortex-M4 Microcontroller Design for IoT Software-Defined Radios. *IEEE Open Journal of Circuits and Systems* online. Piscataway, NJ: IEEE, Červen 2023, sv. 4, s. 123–134. ISSN 2644-1225. Dostupné z: https://doi.org/10.1109/OJCAS.2023.3270752. [cit. 2024-12-08].

[17] Zephyr Project. *Power Management in Zephyr* online. Latest. Open Source Community, leden 2023. Dostupné z: https://docs.zephyrproject.org/latest/services/pm/index.html. [cit. 2024-12-14].

Příloha A

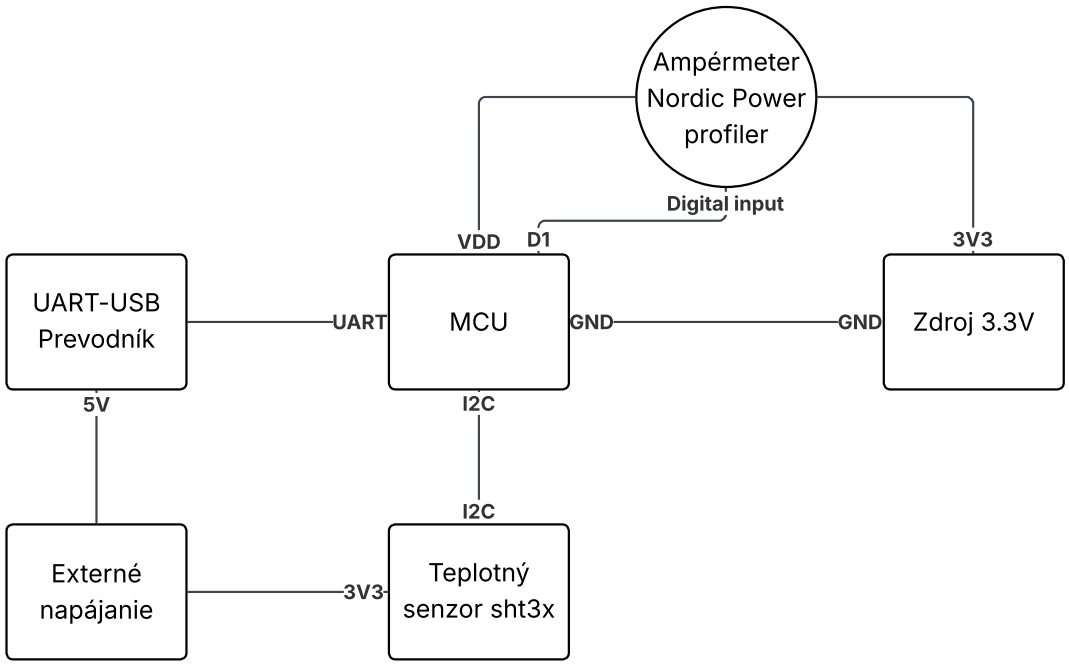
Zoznam testov ULP Peripheral

Slot	ADC	PWM	SPI	RTC
1	Počet vzoriek: 64 Rýchlosť konverzie: 1 kHz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 10%, pevná Počet pulzov: 20		Nastaviť a spustiť časovač
2	Počet vzoriek: 64 Rýchlosť konverzie: 1 kHz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 20%, rastúca Počet pulzov: 40		
3	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 30%, pevná Počet pulzov: 40		
4	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 40%, pevná Počet pulzov: 100	Odoslať 128 B	
5	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 50%, pevná Počet pulzov: 100	Skontrolovať posledný Rx Odoslať 128 B	
6	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 60%, pevná Počet pulzov: 100	Skontrolovať posledný Rx Odoslať 128 B	
7	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 70%, pevná Počet pulzov: 100	Skontrolovať posledný Rx Odoslať 128 B	
8	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 32,786 Hz Perióda: 255 Strieda: 80%, pevná Počet pulzov: 100	Skontrolovať posledný Rx Odoslať 128 B	
9	Počet vzoriek: 1 Rýchlosť konverzie: 1 Hz	Frekvencia: 1 MHz Perióda: 10,000 Strieda: 10%, rastúca Počet pulzov: 30	Skontrolovať posledný Rx Odoslať 128 B	
10	Vypnuté Skontrolovať namerané dáta	Vypnuté	Skontrolovať posledný Rx	Zastaviť a skontrolovať

Tabulka A.1: Sekvencia úloh vykonávaných v teste ULPMark Peripheral.
Zdroj: EEMBC (2023) [3]

Příloha B

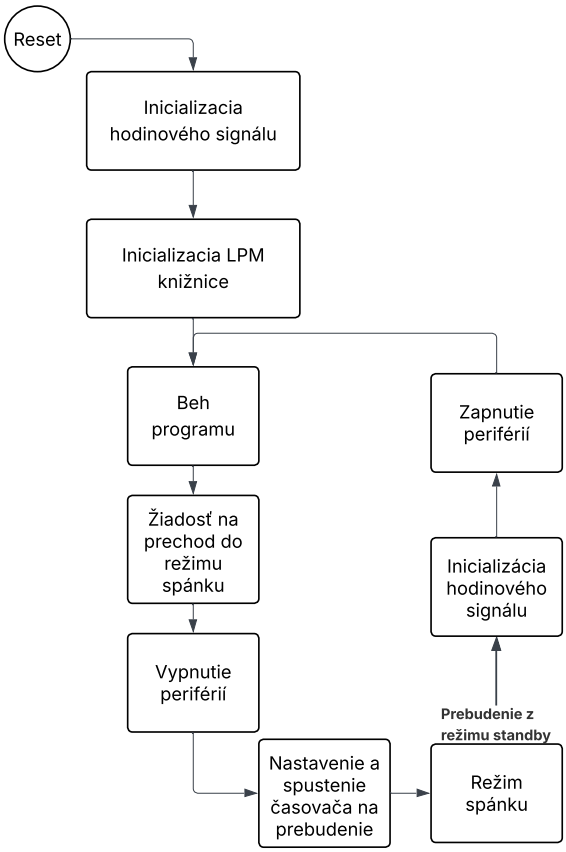
Bloková schéma zapojenia mikrokontrolérov



Obrázek B.1: Zapojenie mikrokontroléru, ampérmetra a senzoru teploty počas merania spotreby

Příloha C

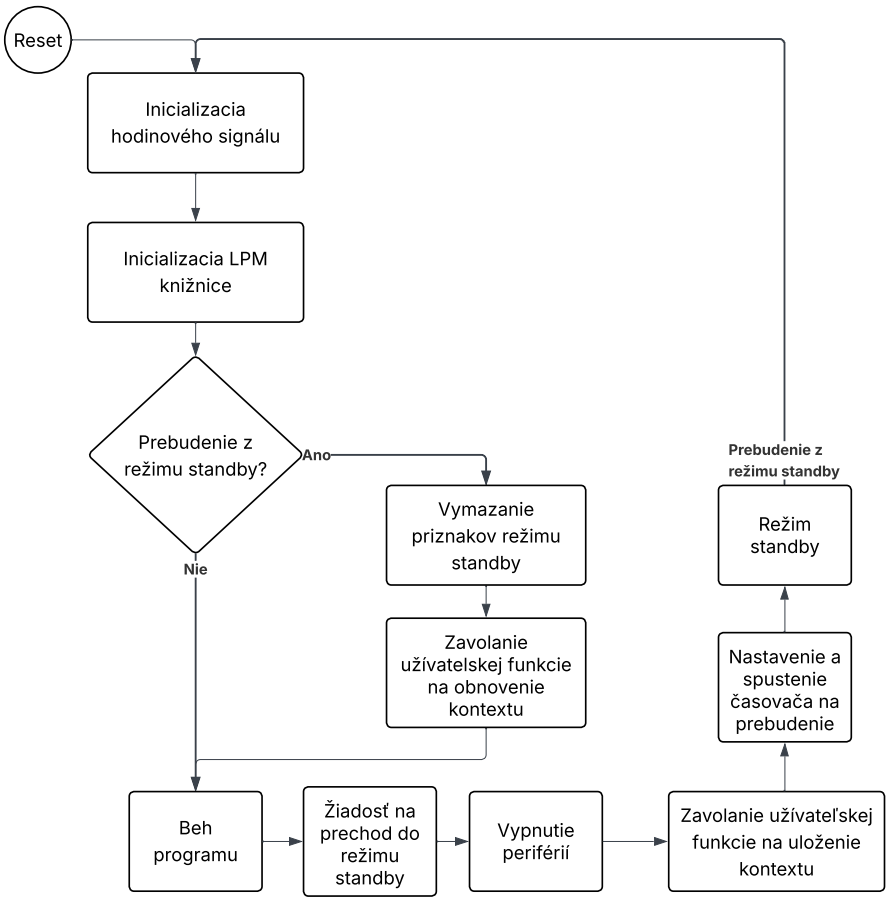
Sekvenčný diagram prebudenia z režimu spánku



Obrázek C.1: Diagram popisujúci kroky potrebné na prechod do režimu spánku a následné prebudenie.

Příloha D

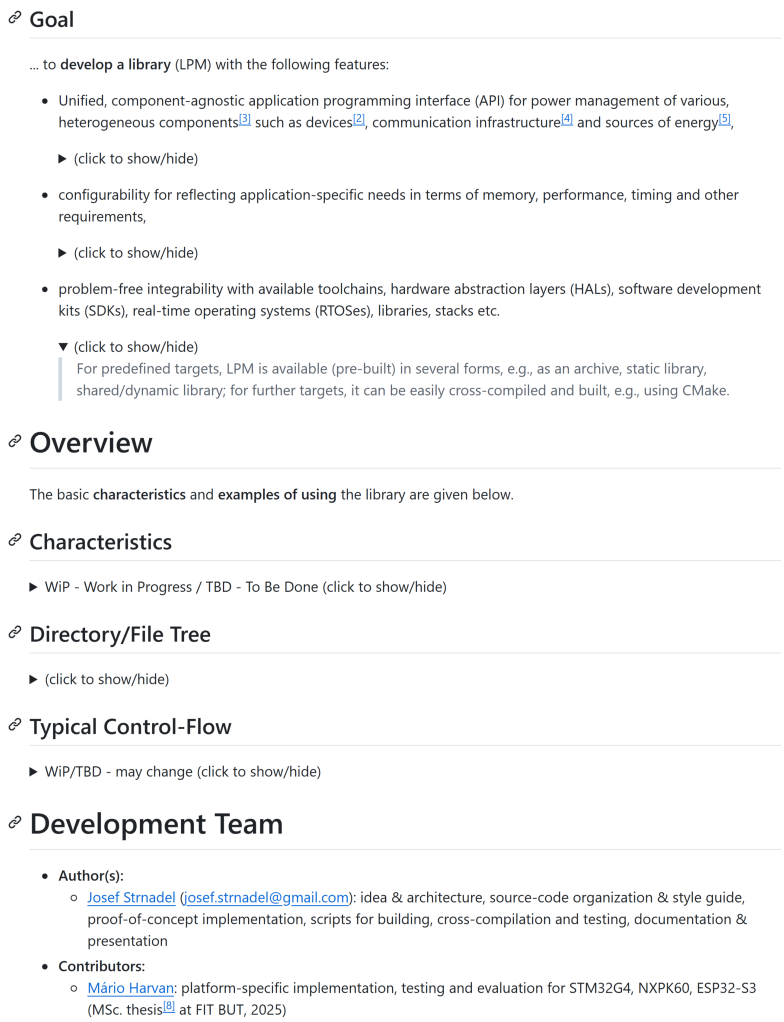
Sekvenčný diagram prebudenia z režimu Standby



Obrázek D.1: Diagram popisujúci kroky potrebné na prechod do režimu standby a následné prebudenie.

Příloha E

Ukážka git repozitáru knižnice LPM



Goal

... to **develop a library** (LPM) with the following features:

- Unified, component-agnostic application programming interface (API) for power management of various, heterogeneous components[3] such as devices[2], communication infrastructure[4] and sources of energy[5],

  ▶ (click to show/hide)

- configurability for reflecting application-specific needs in terms of memory, performance, timing and other requirements,

  ▶ (click to show/hide)

- problem-free integrability with available toolchains, hardware abstraction layers (HALs), software development kits (SDKs), real-time operating systems (RTOSes), libraries, stacks etc.

  ▼ (click to show/hide)

  > For predefined targets, LPM is available (pre-built) in several forms, e.g., as an archive, static library, shared/dynamic library; for further targets, it can be easily cross-compiled and built, e.g., using CMake.

Overview

The basic **characteristics** and **examples of using** the library are given below.

Characteristics

▶ WiP - Work in Progress / TBD - To Be Done (click to show/hide)

Directory/File Tree

▶ (click to show/hide)

Typical Control-Flow

▶ WiP/TBD - may change (click to show/hide)

Development Team

- **Author(s):**
  - Josef Strnadel (josef.strnadel@gmail.com): idea & architecture, source-code organization & style guide, proof-of-concept implementation, scripts for building, cross-compilation and testing, documentation & presentation
- **Contributors:**
  - Mário Harvan: platform-specific implementation, testing and evaluation for STM32G4, NXPK60, ESP32-S3 (MSc. thesis[8] at FIT BUT, 2025)

Obrázek E.1: Snímka obrazovky privátneho git repozitára knižnice LPM [13].

Příloha F

Adresárová štruktúra knižnice LPM

```
lpm/
|__ src/
|   |__ port/
|   |   \__ hw/
|   |       |__ lpm_hwDefs.h       (prevzaté zo šablóny LPM, rozšírené
v rámci diplomovej práce)
|   |       |__ lpm_hwDriver.cc    (prevzaté zo šablóny LPM, rozšírené
v rámci diplomovej práce)
|   |       |__ lpm_hwDriver.h     (prevzaté zo šablóny LPM, rozšírené
v rámci diplomovej práce)
|   |       |__ lpm_hwESP32S3.c    (vytvorené v rámci diplomovej práce)
|   |       |__ lpm_hwESP32S3.h    (vytvorené v rámci diplomovej práce)
|   |       |__ lpm_hwNXP60.c      (vytvorené v rámci diplomovej práce)
|   |       |__ lpm_hwNXP60.h      (vytvorené v rámci diplomovej práce)
|   |       |__ lpm_hwProbe.cc     (prevzaté zo šablóny LPM)
|   |       |__ lpm_hwProbe.h      (prevzaté zo šablóny LPM)
|   |       |__ lpm_hwSTM32G4.c    (vytvorené v rámci diplomovej práce)
|   |       \__ lpm_hwSTM32G4.h    (vytvorené v rámci diplomovej práce)
|   |
|   |__ lpm_common.cc     (prevzaté zo šablóny LPM)
|   |__ lpm_common.h      (prevzaté zo šablóny LPM)
|   |__ lpm_driver.cc     (prevzaté zo šablóny LPM, pridané funkcie)
|   |__ lpm_driver.h      (prevzaté zo šablóny LPM, pridané funkcie)
|   |__ lpm_includes.h    (prevzaté zo šablóny LPM)
|   \__ lpm_misc.h        (prevzaté zo šablóny LPM)
|
|__ CMakeLists.txt         (prevzaté zo šablóny LPM)
|__ CMakePresets.json      (prevzaté zo šablóny LPM)
\__ lpm_config.h           (prevzaté zo šablóny LPM, upravené konfigurácie)
```

Príloha G

Výpis dát z UART zbernice počas spustených testovacích sekvencií

G.1 Testovacia úloha 1.

```
Starting Test 1...
                       ..
          .            ..
       . ..   ....       .
       ..#  ....+#..
    . .    ....+#+... .
    ...   .....+#*...
   ...  .....+######++. .
  .    .....+########+. ..
    ...+....########+.. .
 +.....##++##########.. .
 .....+###*#########+.. .
####################... .
 ...*++###*########+.. .
 +.....##++##########.. .
    ...+....########+.. .
  .    .....+########+. ..
   ...  .....+######*+. .
    ...   .....+#*...
    . .    ....+#+... .
       ..#  ..+.+#..
       . ..   ....       .
          .            ..
Calculation completed in 549 ms
Entering light sleep...
Test 1 completed
```

G.1.1 Testovacia úloha 2.

```
Starting Test 2...
Start temperature measurement
Entering light sleep...
Read temperature: 23.92 °C
GPIO cycle completed
Fibonacci calculation completed in 90 ms, number: 67650000
Entering deep sleep...
Test 2 completed
```

G.1.2 Testovacia úloha 3.

```
Starting Test 3...
Temperature: 24.07 °C
Entering standby mode...

BOOT completed in 90 ms
```

Příloha H

Zoznam vykonaných zmien v knižnici LPM

```
lpm-lite/lpm_config.h                |   3 ++
lpm-lite/src/lpm_common.h            |  17 ++++++-
lpm-lite/src/lpm_driver.h            |   2 +-
lpm-lite/src/lpm_includes.h          |   9 ++--
lpm-lite/src/port/hw/lpm_hwDefs.h    | 103
↪  ++++++++++++++++++++--------------------
lpm-lite/src/port/hw/lpm_hwDriver.cc |  98
↪  +++++++++++++++++++++++++++++++++++++-
lpm-lite/src/port/hw/lpm_hwDriver.h  | 169
↪  ++++++++++++++++++++++++++++++++++++++++-------------------------
lpm-lite/src/port/hw/lpm_hwESP32S3.c | 132
↪  +++++++++++++++++++++++++++++++++++++++++++++++++++
lpm-lite/src/port/hw/lpm_hwESP32S3.h | 157
↪  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
lpm-lite/src/port/hw/lpm_hwNXPK60.c  | 424
↪  +++++++++++++++++++++++++++++++++++++...
lpm-lite/src/port/hw/lpm_hwNXPK60.cc |  11 -----
lpm-lite/src/port/hw/lpm_hwNXPK60.h  | 161
↪  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----
lpm-lite/src/port/hw/lpm_hwProbe.cc  |   4 +-
lpm-lite/src/port/hw/lpm_hwProbe.h   |  21 +++++---
lpm-lite/src/port/hw/lpm_hwSTM32G4.c | 251
↪  +++++++++++++++++++++++++++++++++++++...
lpm-lite/src/port/hw/lpm_hwSTM32G4.h | 239
↪  +++++++++++++++++++++++++++++++++++++...
18 files changed, 1657 insertions(+), 327 deletions(-)
```

Příloha I

Zoznam odovzdaných súborov

```
 /example/
|__ esp32-s3/   - ukážka použitia knižnice a testových úloh na MCU ESP32
|   |__ CMakeLists.txt
|   |__ main.c  - inicializácia, obsluha periférii a spúšťanie testov
|   |__ temp.c  - komunikácia s teplotným senzorom
|   \__ temp.h  - komunikácia s teplotným senzorom
|__ stm32g4/    - ukážka použitia knižnice a testových úloh na MCU STM32G4
|   |__ Core/
|   |   |__ Inc/
|   |   |   |__ main.h
|   |   |   |__ stm32g4xx_hal_conf.h
|   |   |   \__ stm32g4xx_it.h
|   |   \__ Src/
|   |       |__ main.c  - inicializácia, obsluha periférii a testové úlohy
|   |       |__ temp.c  - komunikácia s teplotným senzorom
|   |       \__ temp.h  - komunikácia s teplotným senzorom
|   |__ CMakeLists.txt
|   |__ STM32G474RETX_FLASH.ld
|   |__ STM32G474RETX_RAM.ld
|   |__ st_nucleo_g4.cfg
|   \__ test.ioc    - konfiguračný súbor s nastaveniami mikrokontroléra
|__ nxpk60/         - ukážka použitia knižnice a testových úloh na ESP32
|   \__ source/
|       |__ main.c  - inicializácia, obsluha periférii a testové úlohy
|       |__ syscalls.c
|       |__ temp.c  - komunikácia s teplotným senzorom
|       \__ temp.h  - komunikácia s teplotným senzorom
/lpm_library/       - implementácia knižnice na riadenie spotreby MCU
|__ lpm_hwESP32S3.c - implementácia obslužných funkcií pre ESP32-S3
|__ lpm_hwESP32S3.h - definície režimov spánku a periférií pre ESP32-S3
|__ lpm_hwNXP60.c   - implementácia obslužných funkcií pre NXP K60
|__ lpm_hwNXP60.h   - definície režimov spánku a periférií pre NXP K60
|__ lpm_hwSTM32G4.c - implementácia obslužných funkcií pre STM32G4
|__ lpm_hwSTM32G4.h - definície režimov spánku a periférií pre STM32G4
/low_power_test/    - Testovacie sady určené na porovnanie spotreby MCU
```

```
|__ CMakeLists.txt
|__ lp_test_1.c     - testovacia úloha 1.
|__ lp_test_2.c     - testovacia úloha 2.
|__ lp_test_3.c     - testovacia úloha 3.
|__ lp_tests.h      - definície testovacích funkcií a rozhranie testov
|__ mandelbrot_computation.c - pomocné súbory obsahujúce náročné úlohy
\__ mandelbrot_computation.h - pomocné súbory obsahujúce výpočetné úlohy
/docs/              - Vygenerovaná dokumentácia knižnice
/bin/               - Preložené binárne súbory pre jednotlivé platformy
```

Příloha J

Návod na preklad a spustenie ukážok

J.1 Preklad testov pre mikrokontrolér STM32G4

1. Vyžiadať si prístup k zdrojovým kódom z repozitáru knižnice LPM [13].
2. Skopírovať z odovzdaných súborov z adresáru `lpm\_library` do stiahnutých zdrojových kódov z repozitáru LPM.
3. Prekopírovať skompletizovanú knižnicu LPM do adresáru `example/stm32g4/drivers`.
4. Vygenerovať HAL ovládače pomocou nástroja STM32CUBEMX.
5. Preložiť projekt pomocou STM32CUBEIDE, alebo iného editora podporujúceho mikrokontroléry STM32.
6. Nahrať preložený program na mikrokontrolér pomocou programátora ST ST-Link.

J.2 Preklad testov pre mikrokontroléry ESP32-S3

1. Vyžiadať si prístup k zdrojovým kódom z repozitáru knižnice LPM [13].
2. Skopírovať z odovzdaných súborov z adresáru `lpm\_library` do stiahnutých zdrojových kódov z repozitáru LPM.
3. Prekopírovať skompletizovanú knižnicu LPM do adresáru `example/esp32-s3/libraries`.
4. Importovať projekt do ESP-IDF prostredia, napríklad editor VS Code s doplnkom ESP-IDF.
5. Preložiť projekt pomocou VS Code a doplnku ESP-IDF.
6. Nahrať preložený program na mikrokontrolér pomocou USB rozhrania.

J.3 Preklad testov pre mikrokontroléry NXP K60

1. Vyžiadať si prístup k zdrojovým kódom z repozitáru knižnice LPM [13].

2. Skopírovať z odovzdaných súborov v adresári `lpm\_library` do stiahnutých zdrojových kódov z repozitáru LPM.
3. Prekopírovať skompletizovanú knižnicu LPM do adresáru `example/nxpk60/libraries`.
4. Importovať projekt do MCUXpresso integrovaného prostredia.
5. Preložiť projekt pomocou MCUXpresso.
6. Nahrať preložený program na mikrokontrolér pomocou programátora.

Příloha K

Ukážka dokumentácie knižnice

STM32G4 Hardware Definitions

Definitions specific to STM32G4 hardware for libPwrMan. More...

Typedefs

typedef enum eDevice	**eDevice_t** Supported device IDs.
typedef enum ePwrMode	**ePwrMode_t** Supported power modes.
typedef enum eVoltage	**eVoltage_t** Voltage levels in Volts.
typedef enum eFrequency	**eFrequency_t** Frequency levels in Hz.
typedef enum eSysId	**eSysId_t** System identifiers.

Enumerations

enum **eDevice** { **DEV_STM32_G4_ARMv7M** = 0 }
Supported device IDs. More...

enum **ePwrMode** {
PM_RUN = 0 , **PM_SLEEP** = 1 , **PM_STOP0** = 2 , **PM_STOP1** = 3 ,
PM_STOP2 = 4 , **PM_STANDBY** = 5 , **PM_SHUTDOWN** = 6
}
Supported power modes. More...

enum **eVoltage** { **VDD_1V2** = 0 , **VDD_1V8** = 1 , **VDD_2V1** = 2 , **VDD_3V3** = 3 }
Voltage levels in Volts. More...

enum **eFrequency** {
FREQ_1M = 0 , **FREQ_16M** = 1 , **FREQ_32M** = 2 , **FREQ_64M** = 3 ,
FREQ_170M = 4
}
Frequency levels in Hz. More...

enum **eSysId** { **SYSID_TOPLEVEL** = 0 , **SYSID_COMPONENT** = 1 , **SYSID_SYS** = 2 }
System identifiers. More...

Functions

Obrázek K.1: Ukážka dokumentácie knižnice vytvorenej pomocou nástroja Doxygen.