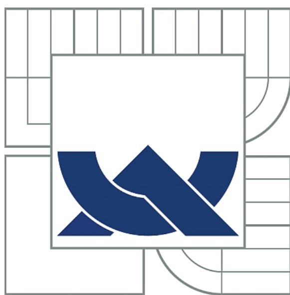
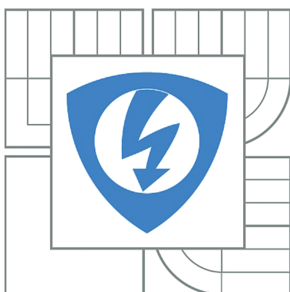


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE RTOS DO MIKROKONTROLÉRŮ STM32 S JÁDREM ARM CORTEX-M4F

IMPLEMENTATION OF RTOS INTO STM32 MICROCONTROLLERS WITH ARM CORTEX-M4F CORE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

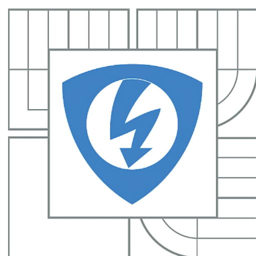
AUTOR PRÁCE — Bc. ADOLF GOTHARD

AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MACHO, Ph.D.

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Adolf Gothard	***ID:***	125432
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Implementace RTOS do mikrokontrolérů STM32 s jádrem ARM Cortex-M4F

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s mikrokontroléry STM32F407VGT6 s jádrem ARM Cortex-M4F a vývojovou deskou STM32F4DISCOVERY.
2. Vyhledejte operační systémy (OS) reálného času, které lze provozovat na mikrokontrolérech s jádrem ARM Cortex-M4F. Popište základní vlastnosti a prostředky těchto OS.
3. Rozeberte možnosti implementace PSD regulátorů do mikrokontroléru STM32F407VGT6 s využitím OS reálného času.
4. Implementujte dva vybrané volně šiřitelné OS reálného času do mikrokontroléru STM32F407VGT6.
5. Realizujte softwarovou implementaci PSD regulátoru při použití vybraných OS reálného času. Řešte i řízení A/D a D/A převodníků.
6. Vyhodnoťte vhodnost vybraných OS reálného času pro realizaci embedded systémů zahrnujících regulační algoritmy.

DOPORUČENÁ LITERATURA:

Mikroprocesory s architekturou ARM On line. Dostupné z http://www.root.cz/clanky/mikroprocesory-s-architekturou-arm/.

Termín zadání:	10.2.2014	***Termín odevzdání:***	19.5.2014

Vedoucí práce: Ing. Tomáš Macho, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Tato diplomová práce se zabývá výběrem a implementací dvou volně šiřitelných operačních systémů reálného času do výkonného 32-bitového mikrokontroléru s jádrem ARM Cortex-M4F. Nejprve je v krátkosti obecně popsána architektura ARM, její programátorský model, instrukční soubor a stručně také jádro Cortex-M4F. Následuje popis architektury použitého mikrokontroléru STM32F407VGT6 od výrobce ST Microelectronics, popis organizace vestavěných pamětí a funkcí vestavěných A/D a D/A převodníků. Další část práce je pak věnována vyhledání operačních systémů reálného času s podporou jádra ARM Cortex-M4F a výběru dvou z těchto systémů pro implementaci. Vybrané systémy jsou podrobněji popsány v následujících dvou kapitolách. Další kapitola se zabývá rozborem možností implementace číslicového PSD regulátoru včetně komplexnějšího systému takových regulátorů při použití operačního systému reálného času. Následuje popis implementace vybraných operačních systémů a navržených regulátorů. V závěru práce je provedeno vyhodnocení vlastností vybraných operačních systémů reálného času z hlediska vhodnosti pro realizaci *embedded* systémů s důrazem na regulační systémy.

Klíčová slova

RTOS, implementace, ARM Cortex-M4F, STM32F407VGT6, PSD regulátor, *embedded* systémy, FreeRTOS, ChibiOS/RT

Abstract

This masters's thesis deals with choice and implementation of two free real-time operating systems into powerful 32-bit microcontroller with ARM Cortex-M4F core. First, there is shortly described the ARM architecture in general, its programmer's model, instruction set and Cortex-M4F core in brief. Next is description of the architecture of used microcontroller STM32F407VGT6 from ST Microelectronics, description of its integrated memories and their organization and functions of its integrated A/D and D/A converters. Next part of this thesis deals with searching real-time operating systems with ARM Cortex-M4F core support and then choose two of these systems for the implementation. The chosen operating systems are more closely described in two following chapters. Next chapter analyses possible implementations of the digital PSD controller and more complex system of such controllers using real-time operating system. Following chapter describes implementation of chosen operating systems and designed controllers. Last chapter deals with evaluation of features and qualities of the chosen real-time operating systems for implementation of embedded control system.

Keywords

RTOS, implementation, ARM Cortex-M4F, STM32F407VGT6, PSD controller, embedded systems, FreeRTOS, ChibiOS/RT

Bibliografická citace:

GOTHARD, A. *Implementace RTOS do mikrokontrolérů STM32 s jádrem ARM Cortex-M4F*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 105 s. Vedoucí diplomové práce Ing. Tomáš Macho, Ph.D..

Prohlášení

Prohlašuji, že svou diplomovou práci na téma *Implementace RTOS do mikrokontrolérů STM32 s jádrem ARM Cortex-M4F* jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **19. května 2014** …………………………

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Tomáši Machovi, Ph.D. za vedení mé práce a dále za odbornou pomoc a rady při jejím vypracování.

V Brně dne: **19. května 2014** ....

podpis autora

Obsah

1 Úvod ... 13

2 Architektura ARM ... 15

2.1 Programátorský model ... 15

2.1.1 Módy procesoru ... 15

2.1.2 Registry ... 16

2.1.3 Výjimky ... 17

2.2 Instrukční soubor ... 18

2.2.1 Instrukční soubor ARM ... 18

2.2.2 Instrukční soubor Thumb ... 18

2.3 Jádro ARM Cortex-M4F ... 19

2.3.1 Standard CMSIS ... 19

3 Mikrokontrolér STM32F407VGT6 ... 21

3.1 Systémová architektura ... 23

3.1.1 Popis jednotlivých sběrnic ... 24

3.2 Organizace paměti ... 25

3.2.1 Vestavěná paměť SRAM ... 25

3.3 Vestavěné A/D převodníky ... 25

3.3.1 Výběr kanálů ... 26

3.3.2 Módy převodu ... 26

3.3.3 Výsledek převodu ... 27

3.3.4 Parametry A/D převodu ... 27

3.4 Vestavěné D/A převodníky ... 27

3.4.1 Parametry D/A převodníků ... 28

3.5 Vývojová deska STM32F4-Discovery ... 28

4 Výběr operačních systémů reálného času ... 30

4.1 Operační systémy reálného času ... 30

4.2 Výběr RTOS s podporou jádra ARM Cortex-M4F ... 31

4.3 Popis některých nalezených operačních systémů ... 32

4.3.1 NuttX ... 32

4.3.2 μC/OS-II ... 34

4.3.3 eCos ... 34

5 Operační systém FreeRTOS 37
5.1 Podporované architektury a nástroje 37
5.2 Struktura zdrojových souborů 38
5.3 Správa úloh 40
5.4 Plánování a priority 42
5.5 Synchronizace a komunikace 44
5.5.1 Fronty 44
5.5.2 Binární semafory 45
5.5.3 Čítací semafory 46
5.5.4 Mutexy 46
5.5.5 Příznaky událostí 47
5.6 Časování 47
5.7 Správa paměti 48
5.7.1 Podpora MPU 50
5.8 Koprogramy 50
6 Operační systém ChibiOS/RT 51
6.1 Podporované architektury a nástroje 52
6.2 Architektura systému 53
6.3 Struktura zdrojových souborů 55
6.4 Správa úloh 56
6.5 Plánování a priority 59
6.6 Synchronizace a komunikace 60
6.6.1 Čítací semafory 61
6.6.2 Binární semafory 61
6.6.3 Mutexy 62
6.6.4 Podmíněné proměnné 63
6.6.5 Příznaky událostí 64
6.6.6 Zprávy 64
6.6.7 Fronty 65
6.7 Správa paměti 65
7 Možnosti implementace PSD regulátorů s využitím OS reálného času 67
7.1 PSD regulátor 67
7.1.1 Vstupy a výstupy 68
7.1.2 Výpočet 69
7.1.3 Další praktické požadavky 70

7.2 Komplexní regulační systémy 71
7.2.1 Systémy s jedinou vzorkovací periodou 71
7.2.2 Systémy s více vzorkovacími periodami 72
7.3 Návrh PSD regulátoru jako úlohy operačního systému 75
8 Implementace 77
8.1 Vytvoření projektu pro systém FreeRTOS 78
8.2 Vytvoření projektu pro systém ChibiOS/RT 79
8.3 Implementace PSD regulátoru v systému FreeRTOS 81
8.4 Implementace PSD regulátoru v systému ChibiOS/RT 84
9 Vyhodnocení vlastností vybraných OS reálného času 87
9.1 Vyhodnocení vlastností systému FreeRTOS 88
9.2 Vyhodnocení vlastností systému ChibiOS/RT 89
10 Závěr 92

Seznam obrázků

Obr. 2-1: Organizace registrů procesoru ARM [1] ... 17
Obr. 2-2: Blokové schéma procesoru s jádrem ARM Cortex-M4(F) [9] ... 20
Obr. 3-1: Blokové schéma mikrokontroléru rodiny STM32F40x [12] ... 22
Obr. 3-2: Architektura sběrnic mikrokontrolérů rodiny STM32F40x [13] ... 23
Obr. 3-3: Blokové schéma zapojení vývojové desky STM32F4-Discovery [14] ... 29
Obr. 5-1: Struktura distribuce operačního systému FreeRTOS [55] ... 39
Obr. 5-2: Přechodový diagram stavů úloh v systému FreeRTOS [60] ... 42
Obr. 5-3: Znázornění spouštění úloh pomocí algoritmu Round-Robin [60] ... 43
Obr. 5-4: Vzájemný vztah mezi aplikací a obsluhou softwarového časovače [74] ... 47
Obr. 6-1: Vztahy jednotlivých součástí operačního systému ChibiOS/RT [89] ... 53
Obr. 6-2: Vztahy mezi subsystémy jádra systému ChibiOS/RT [89] ... 54
Obr. 6-3: Struktura běžného ovladače zařízení [89] ... 55
Obr. 6-4: Pracovní oblast úlohy [101] ... 58
Obr. 6-5: Přechodový diagram stavů úloh v systému ChibiOS/RT [101] ... 58
Obr. 6-6: Seznam připravených úloh [101] ... 59
Obr. 6-7: Vztahy mezi synchronizačními prostředky [89] ... 60
Obr. 6-8: Přechodový diagram stavů čítacího semaforu [104] ... 61
Obr. 6-9: Přechodový diagram stavů binárního semaforu [104] ... 62
Obr. 6-10: Přechodový diagram stavů mutexu [104] ... 63
Obr. 6-11: Správa paměti v systému ChibiOS/RT [89] ... 66
Obr. 7-1: Cyklické provádění řídicího programu s periodou T [121] ... 70
Obr. 7-2: Srovnání statického dělení kódu, EDF a RM pro systém S_{123} [121] ... 75
Obr. 7-3: Principiální znázornění činnosti úlohy operačního systému ... 76
Obr. 8-1: Zjednodušené znázornění struktury projektu pro systém FreeRTOS ... 79
Obr. 8-2: Zjednodušené znázornění struktury projektu pro systém ChibiOS/RT ... 81
Obr. 8-3: Vývojový diagram úlohy pro PSD regulátor ... 86

Seznam tabulek

Tab. 2-1: Módy procesoru s architekturou ARM [1] 15
Tab. 2-2: Výjimky a módy pro jejich obsluhu [1]..... 18
Tab. 5-1: Struktura TCB v systému FreeRTOS 41
Tab. 6-1: Architektury podporované operačním systémem ChibiOS/RT [83] 52
Tab. 6-2: Struktura reprezentující úlohu v systému ChibiOS/RT..... 57

1 ÚVOD

V současné době jsou číslicové řídicí systémy všeobecně rozšířené pravděpodobně ve všech odvětvích řídicích úloh. Jedním z hlavních důvodů je skutečnost, že veškeré výpočty realizované softwarově lze snadno a rychle modifikovat podle aktuálních potřeb pouze změnou programu řídicího počítače. To dává k dispozici velmi univerzální nástroj, jehož použitelnost pro danou řídicí úlohu je omezena v podstatě jen výkonem samotného počítače nebo mikrokontroléru. Moderní procesory pak disponují výkonem, který je pro většinu řídicích úloh dostatečný. Dalšími výhodami jsou prakticky absolutní stabilita nastavených parametrů, možnost jejich snadnější adaptivní modifikace za běhu a dále např. možnost komunikace a synchronizace s dalšími systémy.

Základní společnou vlastností číslicových řídicích systémů je samotný charakter jejich činnosti, který je diskrétní a sekvenční (nejsou zde uvažovány vícejádrové ani víceprocesorové systémy, popř. systémy s obvody FPGA). Při řízení jednodušších systémů bez zvláštních časových požadavků tato vlastnost nepředstavuje problém. Čím je však řídicí systém komplexnější, tím obtížnější je zaručit požadované časové chování. U časově kritických systémů (tzv. *real-time* systémy) obvykle nesplnění daných časových požadavků způsobí výrazné snížení výkonu nebo i selhání celého systému, což může mít velmi vážné důsledky. Pro *real-time* systémy již tedy může sekvenční zpracování algoritmu představovat problém. Tento problém je obvykle řešen použitím metod, které zpracovávají provádění řídicího algoritmu tak, aby byla v daný okamžik vždy provedena správná operace. Jednou z těchto metod je použití operačního systému reálného času (*real-time operating system*, RTOS).

Tato diplomová práce se zabývá implementací operačních systémů reálného času do moderního výkonného 32-bitového mikrokontroléru s jádrem ARM Cortex-M4F z rodiny STM32 firmy ST Microelectronics a následnou realizací číslicového regulátoru s využitím těchto systémů. Cílem práce je tedy výběr, popis a implementace dvou volně šiřitelných operačních systémů reálného času do daného mikrokontroléru a dále vyhodnocení jejich použitelnosti pro realizaci regulačních systémů.

Na začátku práce je proveden stručný obecný popis procesorové architektury ARM a velmi krátce je popsáno také jádro Cortex-M4F, na kterém je založen použitý mikrokontrolér STM32F407VGT6, jehož popisu se věnuje třetí kapitola. Je zde popsána architektura mikrokontroléru, organizace paměti a podrobněji také vestavěné A/D a D/A převodníky, které jsou velmi důležitou součástí řídicích systémů. Dále je uveden krátký popis použité vývojové desky STM32F4-Discovery. Čtvrtá kapitola je věnována jednak stručnému obecnému popisu operačních systémů reálného času, jednak vyhledání a popisu OS reálného času s podporou jádra ARM Cortex-M4F a výběru systémů pro implementaci do použitého mikrokontroléru. Následující dvě kapitoly se pak zabývají podrobnějším popisem vybraných OS reálného času. U těchto kapitol byla pokud možno dodržována stejná struktura.

Sedmá kapitola je věnována rozboru možností implementace číslicového PSD regulátoru. Začíná stručným obecným popisem PSD regulátoru jako číslicové náhrady PID regulátoru, následně jsou probrány nejdůležitější části jeho implementace a také některé další praktické požadavky. Ve druhé části kapitoly jsou probrány možnosti implementace komplexnějších regulačních systémů, které využívají více regulátorů. Poslední část kapitoly je věnována možnostem návrhu regulátoru jako úlohy pro použití v OS reálného času.

V další kapitole je pak popsána již praktická implementace jednak samotných vybraných operačních systémů, jednak navržených úloh pro PSD regulátory s použitím těchto systémů.

Devátá kapitola je věnována poslednímu bodu zadání a je zde provedeno vyhodnocení vlastností vybraných OS reálného času pro realizaci *embedded* systémů, které zahrnují regulační algoritmy.

2 ARCHITEKTURA ARM

Mikroprocesory s architekturou ARM jsou v současné době velmi rozšířené především v mobilních zařízeních. V oblasti mobilních telefonů se podíl ARM procesorů blíží sto procentům. Dále jsou rozšířené v síťových, spotřebních a vestavěných (embedded) zařízeních, kde mají dlouhodobě vysoký podíl zejména díky nízké spotřebě. Také politika firmy ARM Holdings, kdy je jednotlivým výrobcům licencována architektura jako duševní vlastnictví, s tím, že výrobce může na čip přidat další zařízení (paměti, periferie atd.), přispěla k rozšíření procesorů s touto architekturou.

2.1 Programátorský model

Jak bylo uvedeno, ARM je architektura typu RISC (Reduced Instruction Set Computer). Typickou vlastností RISC architektur je malý soubor instrukcí stejné délky, větší množství registrů, jednoduché adresování, zřetězení instrukcí (pipelining). Operace s daty jsou prováděny pouze v registrech (load/store architektura).

Architektura ARM navíc nabízí větší kontrolu nad aritmeticko-logickou jednotkou (ALU), adresní módy s automatickou inkrementací nebo dekrementací, instrukce typu *Load and Store Multiple* pro načítání/ukládání více dat současně a podmíněné provádění instrukcí pro optimalizaci počtu skoků v programu [1]. Procesory ARM pracují s těmito datovými typy:

Byte (8 bitů)
Halfword (16 bitů)
Word (32 bitů).

2.1.1 Módy procesoru

Architektura ARM podporuje sedm procesorových módu. Mezi jednotlivými módy lze přepínat softwarově nebo při příchodu přerušení/ošetření výjimek. Tyto módy jsou uvedeny v Tab. 2-1.

Tab. 2-1: Módy procesoru s architekturou ARM [1]

mód procesoru	zkratka módu	popis
User	usr	Mód pro provádění běžných programů
FIQ	fiq	Obsluha rychlých přerušení
IRQ	irq	Obecná obsluha přerušení
Supervisor	svc	Chráněný mód pro operační systém
Abort	abt	Ochrana paměti/virtuální paměti
Undefined	und	Softwarová emulace hardwarových koprocesorů
System	sys	Pro vykonávání důležitých úloh operačního systému

Většina uživatelských programů běží v módu User. Pokud je procesor v tomto módu, nemá běžící program přístup k některým částem systému a není možná programová změna módu. Ostatní módy jsou privilegované (*privileged*) a mají úplný přístup ke všem součástem systému. V privilegovaných módech lze také libovolně měnit módy procesoru. Z těchto šesti privilegovaných módů je pět módů výjimkových (*exception*). Jsou to:

FIQ
IRQ
Supervisor
Abort
Undefined

Do výjimkových módů se procesor dostává příchodem určité výjimky (např. přerušení). Každý výjimkový mód má přiřazeny některé speciální registry, aby nebyla ovlivňována činnost programu v uživatelském (User) módu (viz část 2.1.2).

Posledním módem je systémový (System) mód. Tento mód má k dispozici stejné registry jako uživatelský (User) mód. Stále se však jedná o privilegovaný mód. Je určen pro některé důležité úlohy operačního systému, které vyžadují plný přístup k systémovým zdrojům a současně není vhodné, aby používaly stejné speciální registry jako výjimkové módy. Stav úlohy, běžící v módu System, není tedy narušen příchodem nějaké výjimky.

2.1.2 Registry

Procesory ARM mají celkem 37 registrů o délce 32 bitů. Z toho 31 registrů pro obecné použití včetně programového čítače (*program counter*) a šest stavových (*status*) registrů. Registry jsou organizovány ve skupinách a jejich viditelnost závisí na aktuálním módu procesoru. V každém módu je viditelných 15 registrů pro všeobecné použití (R0 až R14), programový čítač (R15) a jeden nebo dva stavové registry. Ve výjimkových módech jsou vždy některé registry nahrazeny speciálními registry, určenými pouze pro daný mód. Registry R0 až R7 jsou stejné ve všech módech a nemají žádné systémové využití. Mohou být použity k jakémukoliv účelu.

Registr R13 je normálně používán jako ukazatel zásobníku (SP – *stack pointer*), registr R14 (LR – *link register*) slouží k uchování návratových adres z podprogramů a přerušení. Registr R15 slouží jako programový čítač (PC – *program counter*). Pro podrobný popis organizace registrů a jejich funkce viz [1], [2]. Organizace registrů je znázorněna na Obr. 2-1

Obr. 2-1: Organizace registrů procesoru ARM [1]

Modes

Privileged modes (System – Fast interrupt)

Exception modes (Supervisor – Fast interrupt)

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

2.1.3 Výjimky

Architektura ARM podporuje sedm typů výjimek, které mohou být generovány vnitřními nebo vnějšími zdroji. Před obsluhou výjimky je aktuální stav uložen, aby bylo možné po ukončení obsluhy pokračovat ve vykonávání původního programu. Může nastat více výjimek současně. V Tab. 2-2 jsou typy výjimek a módy, ve kterých probíhá obsluha dané výjimky.

Při příchodu výjimky je nejprve uložena návratová adresa do příslušného registru R14 (každý mód má k dispozici vlastní registr R14) a aktuální stav (registr CPSR) je uložen do příslušného registru SPSR. Následně jsou provedena další nastavení v závislosti na daném módu a do programového čítače je uložena adresa vektoru příslušné výjimky.

Po vykonání obsluhy přerušení je obnoven registr CPSR a obsah příslušného registru R14 je přesunut do programového čítače. Pro podrobný popis činnosti a nastavení při obsluze jednotlivých typů výjimek viz [1].

Tab. 2-2: Výjimky a módy pro jejich obsluhu [1]

typ výjimky	mód
Reset	Supervisor
Nedefinované instrukce	Undefined
Softwarové přerušení	Supervisor
Selhání načtení instrukce	Abort
Selhání přístupu do datové paměti	Abort
Přerušení	IRQ
Rychlé přerušení	FIQ

2.2 Instrukční soubor

2.2.1 Instrukční soubor ARM

Původní instrukční soubor procesorů ARM obsahuje instrukce délky 32 bitů, zarovnané v paměti na celá slova, tj. instrukce jsou uloženy na adresách dělitelných 4 (slovo ARM procesorů představují 4 byty).

Téměř každá instrukce obsahuje podmínkové pole, které je umístěno v nejvyšších čtyř bitech. Na obsahu podmínkového pole pak závisí, zda bude instrukce provedena. Tato speciální vlastnost ARM instrukcí umožňuje minimalizovat počet skoků v programu a tím zvýšit výkon.

Další vlastností je zřetězené zpracování instrukcí (*pipeline*). Zpracování instrukce je rozděleno na několik částí a tyto části jsou prováděny jednotlivě. Díky tomu je možné zpracovávat více instrukcí současně v různém stavu rozpracování. V případě architektury ARMv7 a starší jsou instrukce rozděleny na tři fáze a je tedy možné zpracovávat tři instrukce současně. První fází je načtení instrukce (*fetch*), dále dekódování (*decode*) a provedení instrukce (*execute*).

2.2.2 Instrukční soubor Thumb

Novější rodiny procesorů ARM jsou vybaveny instrukční sadou Thumb. Tato instrukční sada vznikla pro použití ve vestavěných (*embedded*) systémech, kde je kromě nízké spotřeby zapotřebí také úspora paměti.

Instrukce Thumb jsou podmnožinou původních instrukcí ARM a mají délku pouze 16 bitů. To umožňuje dosáhnout větší hustoty kódu a většího výkonu u implementací, používajících datové sběrnice o šířce 16 bitů a užší. V [3] je uvedeno, že program, využívající instrukce Thumb, přeložený z kódu napsaného v jazyku C, dosahuje 65 % velikosti programu, který využívá instrukce ARM. Zkrácením instrukcí však došlo k odstranění podmínkového pole a je tedy nutné využívat standardních skoků.

Thumb instrukce využívají standardní registry a pracují s 32-bitovými daty tak jako instrukce ARM. To umožňuje přepínání mezi instrukčními soubory i v rámci jednotlivých funkcí [4]. Před vykonáním Thumb instrukce je tato převedena dekodérem na délku 32 bitů.

Instrukční sada Thumb-2 vznikla rozšířením původní sady Thumb o některé 32-bitové instrukce. Cílem je sloučit výhody instrukčních sad ARM a Thumb, tedy vysoký výkon a vysokou hustotu kódu, tak, aby nebylo nutné přepínat mezi jednotlivými sadami. Podle [5] je úspora velikosti kódu Thumb-2 oproti ARM 26 % a pokles relativního výpočetního výkonu pouze o 2 %.

2.3 Jádro ARM Cortex-M4F

Jádro Cortex-M4 patří do rodiny procesorů Cortex-M. Tato rodina je určena pro použití ve vestavěných (*embedded*) systémech, kde je zapotřebí nízká spotřeba a malé rozměry. Tyto mikrokontroléry mohou mít velmi široké využití (např. automobilová technika, řízení motorů, průmyslová automatizace). Procesor ARM Cortex-M4 je postaven na architektuře ARMv7-M [6], [7] a má implementovánu instrukční sadu Thumb-2. Instrukce jsou rozděleny na tři fáze a jejich zpracovávání je zřetězeno.

Procesor je navržen zejména pro číslicové zpracování signálů. Pro tyto účely má implementovány některé vlastnosti DSP (instrukce MAC, SIMD). Nabízí také jednotku pro zpracování čísel s pohyblivou řádovou čárkou (FPU – *floating point unit*) s přesností *single precision* vyhovující standardu IEEE 754 [8]. Procesor s jednotkou FPU je označován jako Cortex-M4F.

Dále je vestavěna jednotka pro ochranu a správu paměti (MPU – *memory protection unit*). Lze ji použít k ochraně přístupu do paměti v případě provádění kritického kódu, může být ovládána operačním systémem reálného času.

Podrobné informace o vlastnostech a použití procesoru Cortex-M4F lze nalézt v [9]. Na je blokové schéma procesoru s jádrem ARM Cortex-M4(F).

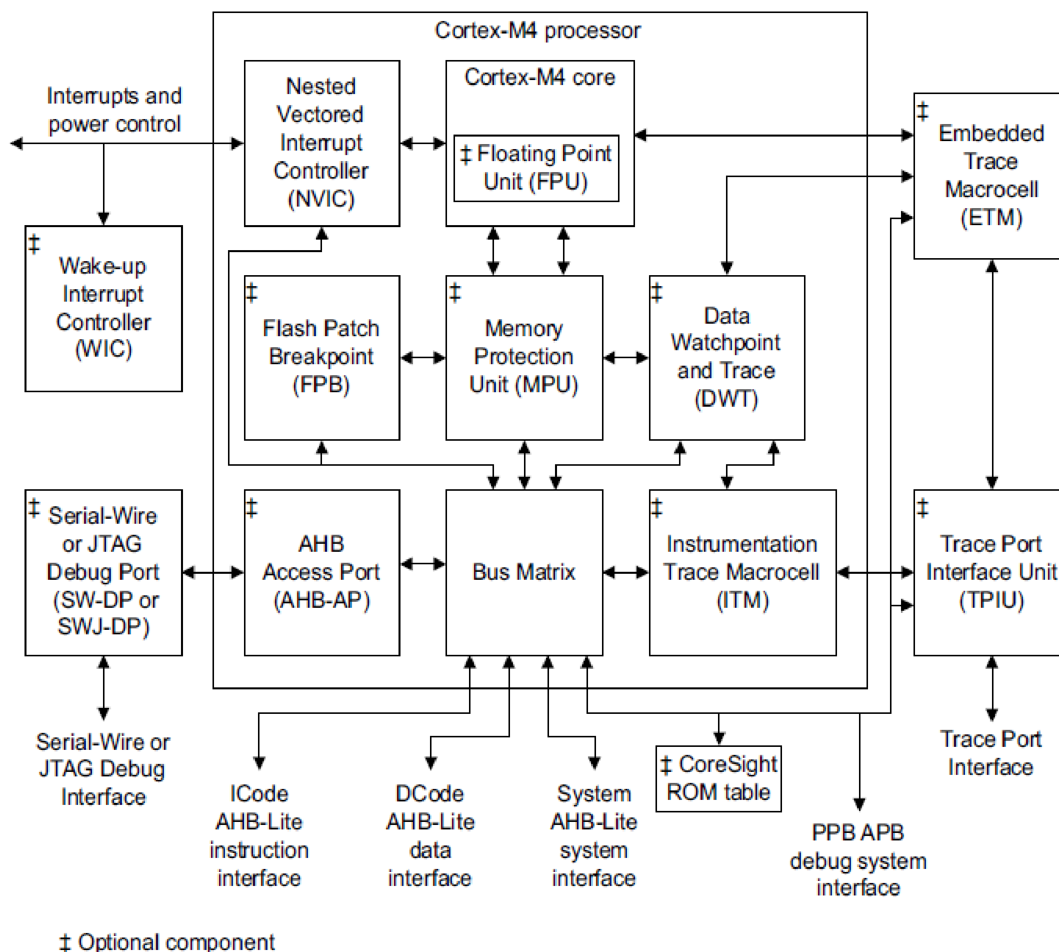
2.3.1 Standard CMSIS

Ve vestavěných systémech je vývoj software jedním z hlavních faktorů, ovlivňujících cenu zařízení. Komplexnost software se zvyšuje a tím dochází i ke zvyšování ceny za jeho vývoj [10]. Proto je snaha o standardizaci a opakované využití softwarových komponent.

Standard CMSIS (*Cortex Microcontroller Software Interface Standard*) je, jak z názvu vyplývá, určen pro mikrokontroléry rodiny Cortex-M. Jedná se o abstraktní rozhraní (HAL – *hardware abstraction layer*) nezávislé na výrobci, popř. rozšiřitelné výrobcem, které zprostředkovává přístup k součástem jádra mikrokontroléru a také

k periferiím. CMSIS definuje aplikační rozhraní (API – *application programming interface*) pro všechny mikrokontroléry Cortex-M. Podrobnější popis standardu CMSIS lze nalézt např. v [10] a [11].

Obr. 2-2: Blokové schéma procesoru s jádrem ARM Cortex-M4(F) [9]



3 MIKROKONTROLÉR STM32F407VGT6

STM32F407VGT6 je výkonný 32-bitový mikrokontrolér firmy ST Microelectronics, který patří do rodiny mikrokontrolérů STM32F40x.

Tato rodina je založena na RISC jádře ARM Cortex-M4F, které lze taktovat až na 168 MHz. Jádro mikrokontroléru obsahuje jednotku pro výpočty s čísly s pohyblivou řádovou čárkou (FPU) s přesností *single precision* (viz část 2.3). Je implementována také sada DSP instrukcí. Díky tomu lze tyto mikrokontroléry použít i pro náročnější zpracování signálu.

Mikrokontrolér STM32F407VGT6 obsahuje rychlé vestavěné paměti. Programová paměť FLASH má velikost 1MB. Pro verifikaci obsahu paměti může být také použita jednotka CRC (*cyclic redundancy check*). Vestavěná systémová paměť SRAM má velikost 192 kB a mikrokontrolér dále obsahuje 4kB záložní SRAM. K této záložní paměti může přistupovat pouze CPU a je chráněna proti nežádoucímu přepisu.

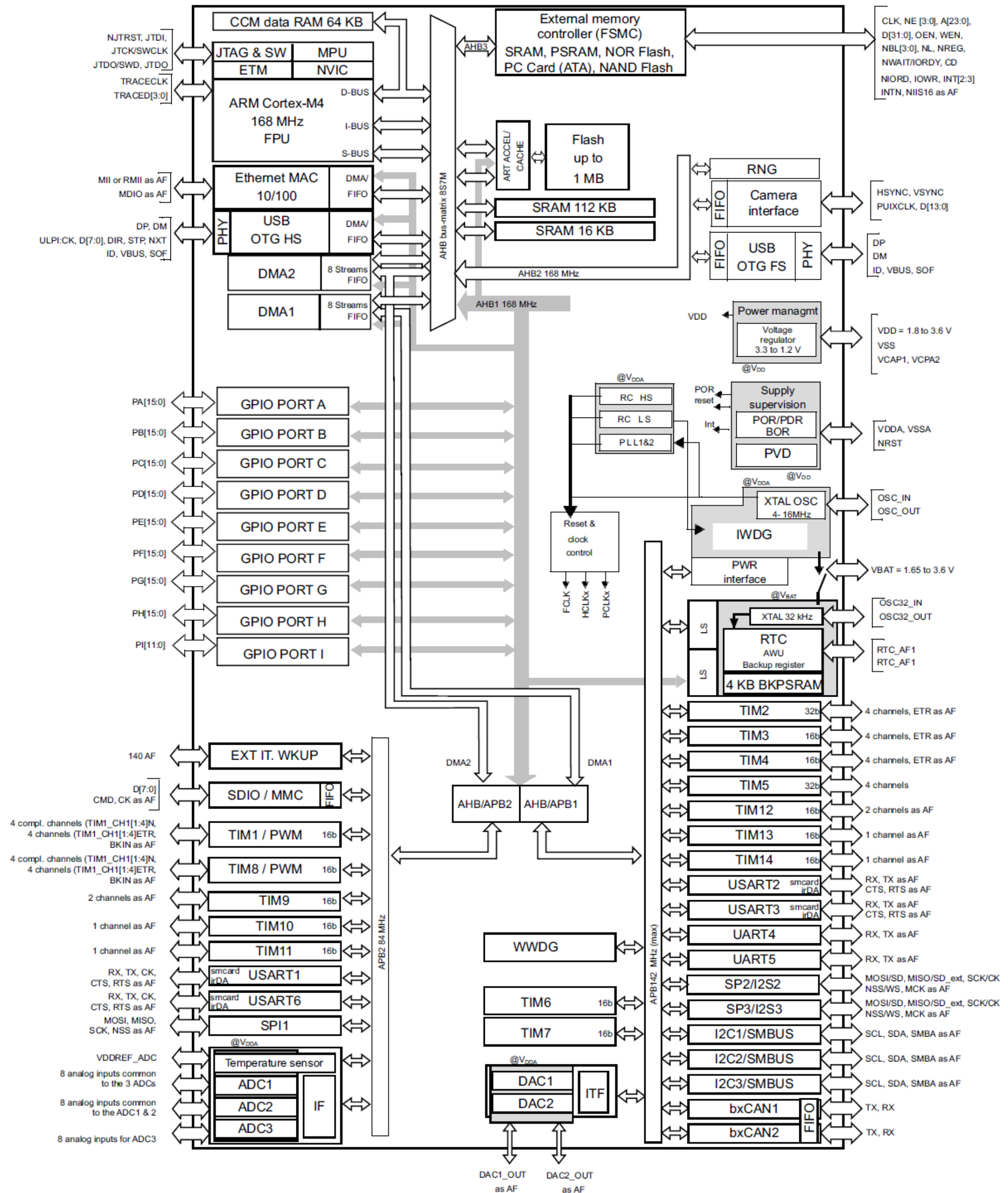
Součástí mikrokontroléru je řada vstupně/výstupních portů a periferií. Těmito periferiemi jsou tři 12-bitové A/D převodníky, dva 12-bitové D/A převodníky, nízkopříkonový obvod reálného času (RTC), dvanáct 16-bitových časovačů včetně dvou PWM časovačů pro řízení motorů, dva 32-bitové časovače, generátor náhodných čísel.

Mezi komunikační rozhraní patří:

3x I^2C
3x SPI
2x I^2S
4x USART
2x UART
USB OTG
2x CAN
SDIO/MMC
Ethernet
Rozhraní pro kameru

Napájecí napětí mikrokontroléru může být 1,8 až 3,6 V. Dodáván je v pouzdře LQFP100 a nabízí 82 vstupně/výstupních vývodů pro obecné použití (GPIO). Na je blokové schéma mikrokontrolérů rodiny STM32F40x. Podrobnější popis mikrokontroléru lze nalézt v [12], [13].

Obr. 3-1: Blokové schéma mikrokontroléru rodiny STM32F40x [12]



3.1 Systémová architektura

Ke vzájemné komunikaci jednotlivých částí slouží vícevrstvý systém propojení několika 32-bitových sběrnic (AHB *bus matrix*). Pomocí tohoto systému je propojeno osm sběrnic typu *master* a sedm typu *slave*. Systém umožňuje současné propojení různých zařízení a tím zrychlení činnosti mikrokontroléru. Také zajišťuje arbitraci při současné komunikaci více zařízení typu master. K arbitraci je použit algoritmus Round-Robin [13].

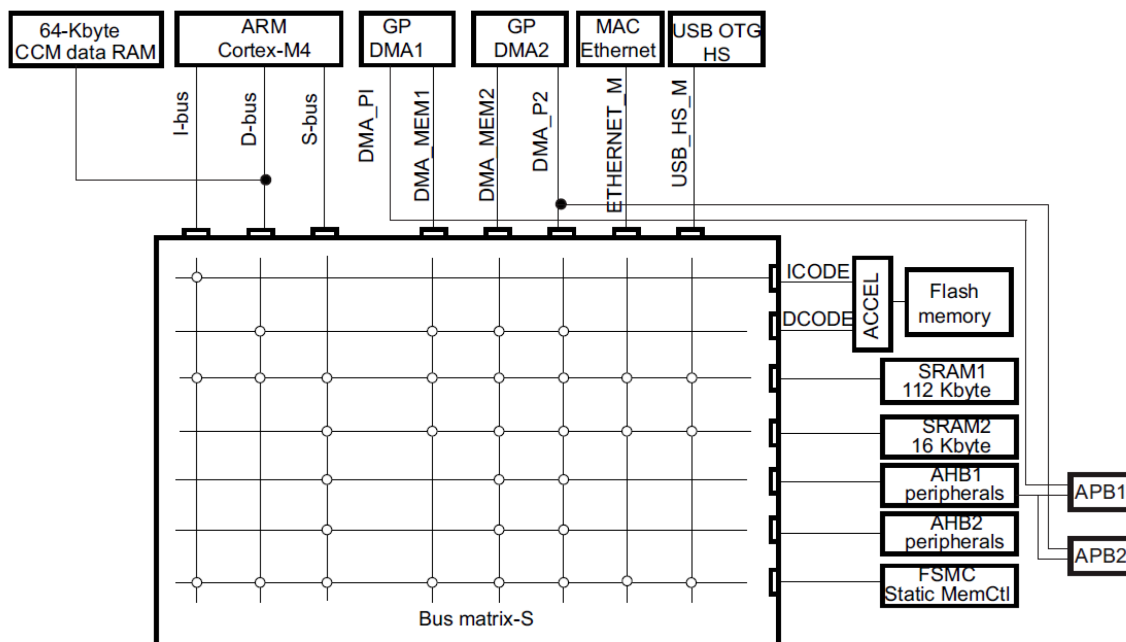
Sběrnice typu *master* jsou:

1) sběrnice I-bus, D-bus a S-bus jádra Cortex-M4F
2) DMA1 paměťová sběnice
3) DMA2 paměťová sběrnice
4) DMA2 sběrnice pro periferie
5) Ethernetová DMA sběrnice
6) USB DMA sběrnice

Sběrnice typu *slave* jsou:

1) ICode sběrnice FLASH paměti
2) DCode sběrnice FLASH paměti
3) SRAM1 (112 kB)
4) SRAM2 (16 kB)
5) Sběrnice periferií AHB1
6) Sběrnice periferií AHB2
7) Paměťové rozhraní FSMC

Obr. 3-2: Architektura sběrnic mikrokontrolérů rodiny STM32F40x [13]



3.1.1 Popis jednotlivých sběrnic

Sběrnice I-Bus

Tato sběrnice slouží k načítání instrukcí z paměti obsahující kód (vestavěné FLASH/SRAM paměti nebo externí paměť přes rozhraní FSMC).

Sběrnice D-bus

Používá se k načítání literálů a pro přístup ladicích prostředků. Dále je použita ke komunikaci jádra s CCM (*core coupled memory*) datovou RAM 64kB, která je přístupná pouze z jádra mikrokontroléru.

Sběrnice S-bus

Systémová sběrnice určená pro přístup jádra mikrokontroléru k paměti SRAM a k periferiím.

Paměťová sběrnice DMA

Tato sběrnice je používána k přímému přístupu do pamětí (mechanismus DMA – *direct memory access*).

Periferní sběrnice DMA

Používána mechanismem DMA pro přímý přístup k periferiím mikrokontroléru nebo k přenosům mezi pamětmi.

Ethernet a USB DMA sběrnice

Sběrnice použité ke přímému přístupu ethernetového a USB OTG zařízení k paměti.

AHP/APB můstky

Součástí systému jsou dva AHB/APB můstky, označené APB1 a APB2, které zajišťují plně synchronní propojení mezi systémovou sběrnicí AHB (*advanced high-performance bus*) a dvěma periferními APB sběrnicemi. K těmto periferním sběrnicím jsou připojeny čítače/časovače, A/D a D/A převodníky a většina komunikačních rozhraní mikrokontroléru.

Můstky umožňují správu periferních hodinových signálů, vhodnou volbu jejich frekvence a případné odpojení hodin pro úsporu energie. Po resetu mikrokontroléru jsou všechny hodinové signály zakázány kromě komunikačních rozhraní pro FLASH a SRAM paměti.

3.2 Organizace paměti

Mikrokontrolér STM32F407VGT6 disponuje lineárním adresovým prostorem o velikosti 4 GB, který je společný pro programovou paměť, datovou paměť, registry a vstupně/výstupní porty. Data jsou kódována ve formátu *little-endian*, tj. nejméně významný byte (LSB) dat je uložen na paměťové místo s nejnižší adresou. Adresový prostor je rozdělen do osmi bloků po 512 MB. Organizace paměti je podrobně popsána v [13].

3.2.1 Vestavěná paměť SRAM

Mikrokontrolér STM32F407VGT6 obsahuje celkem 192 kB vestavěné systémové paměti SRAM a 4 kB záložní paměti SRAM. K paměti SRAM lze přistupovat po bytech, půl-slovech (16 bitů) nebo celých slovech (32 bitů). Systémová paměť SRAM je rozdělena na následující části.

SRAM 1 a SRAM 2

Paměti o velikosti 112 kB a 16 kB přístupné pro všechna zařízení typu master. Díky tomuto rozdělení paměti je možné používat jednotlivé části současně.

CCM – *core coupled memory*

Paměť o velikosti 64 kB. Tato paměť je přístupná pouze z CPU pomocí D-bus sběrnice.

3.3 Vestavěné A/D převodníky

Převod analogové hodnoty na číslo pro zpracování v počítači nebo mikrokontroléru a následný převod čísla zpět na analogovou hodnotu je v číslicové regulační technice jedním z hlavních problémů. Vlastnosti a parametry těchto převodů mají velký vliv na celý regulační obvod. Proto budou v této a v následující podkapitole podrobněji popsány A/D a D/A převodníky vestavěné v mikrokontroléru STM32F407VGT6.

Mikrokontrolér STM32F407VGT6 disponuje třemi vestavěnými A/D převodníky. Tyto převodníky jsou založeny na principu postupné aproximace a mají nastavitelné rozlišení (až 12 bitů). Převodníky mohou využívat 16 externích kanálů a tři interní kanály (vnitřní referenční napětí, napětí baterie a vestavěný snímač teploty). Samozřejmostí je možnost vyvolání přerušení po ukončení převodu. Navíc lze vyvolat přerušení při překročení nebo nedosažení určité úrovně vstupního signálu (*analog watchdog*). Každý kanál může být vzorkován s vlastní vzorkovací frekvencí.

Hodinový signál může být buď společný pro všechny převodníky (odvozený z hodinového signálu pro periferie dělením dvěma, čtyřmi, šesti nebo osmi), nebo lze použít přímo tento hodinový signál. Lze zvlášť povolit pro každý převodník.

Převodníky lze také použít v tzv. *Multi ADC* módech, kde dva nebo tři převodníky pracují společně a umožňují tak dosáhnout kratší doby převodu při práci s více kanály, popř. je možné dosáhnout až třikrát rychlejší vzorkování jednoho kanálu (při použití tří převodníků).

3.3.1 Výběr kanálů

A/D převodníky mají k dispozici dohromady 16 externích multiplexovaných kanálů. Převod těchto kanálů lze rozdělit do dvou skupin.

Skupina *regular group* může být sestavena až z šestnácti převodů (každý kanál je převeden). Počet a pořadí převodů je nutno nastavit. Skupina *injected group* může být složena až ze čtyř převodů. Opět je nutno nastavit počet kanálů a jejich pořadí.

3.3.2 Módy převodu

Převod může být prováděn v následujících módech [13].

Single

A/D převodník vykoná pouze jediný převod. Tento převod může být odstartován programově nebo externím spouštěcím vstupem. Na konci převodu je uložen výsledek a je nastaven příznak ukončení převodu. Je možné vyvolat přerušení.

Continuous

V tomto módu probíhají konverze jako v módu single, ale po provedení převodu je ihned zahájen nový převod. Tento mód nelze použít pro kanály typu *injected.*

Scan

Tento mód slouží k převodu více kanálů. Pro každý kanál, přiřazený do skupiny, je proveden převod tak jako v módu single a následně je automaticky spuštěn převod následujícího kanálu. Je možné ukončit činnost po převodu posledního kanálu, nebo pokračovat podobně jako v módu continuous. Přerušení je možné vyvolat buď po převodu každého kanálu nebo po ukončení převodu celé skupiny.

Také je možné pomocí DMA přenášet výsledek převodu přímo do SRAM. To platí pouze pro kanály typu *regular.*

Discontinuous

Slouží k provedení převodu určitého počtu kanálů (max. 8) z vybrané skupiny pomocí externího spouštěcího vstupu.

3.3.3 Výsledek převodu

Výsledek převodu je uložen do 16-bitového datového registru. Data mohou být zarovnána vlevo nebo vpravo. Pro výsledek převodu skupiny *injected group* lze navíc odečíst uživatelsky definovanou konstantu a získat tak např. záporné číslo.

V případě požadavku rychlé konverze více než jednoho kanálu skupiny *regular group* lze výhodně použít DMA mechanismu k uložení více převedených hodnot do paměti.

Přerušení lze vyvolat po ukončení převodu skupiny *regular group*, po ukončení převodu skupiny *injected group*, dále při překročení/nedosažení určité úrovně vstupního signálu (*analog watchdog*) a při ztrátě dat (přetečení) při přenosu pomocí DMA.

3.3.4 Parametry A/D převodu

Kromě funkčních vlastností A/D převodníku jsou velmi důležité také parametry vlastního převodu jako rychlost, přesnost a rozlišení. Tyto parametry jsou zvláště důležité pro použití v číslicové regulaci.

Jak bylo uvedeno, vestavěné A/D převodníky mikrokontroléru STM32F407VGT6 mají nastavitelné rozlišení 6, 8, 10 nebo 12 bitů.

Podle [12] je celkový čas převodu (včetně doby vzorkování) při rozlišení 12 bitů a hodinové frekvenci převodníku 30 MHz maximálně 16,40 µs. Maximální rychlost vzorkování při stejné frekvenci (30 MHz) a rozlišení 12 bitů je 2 Msps při použití jediného převodníku. V případě použití tří převodníků je to až 6 Msps. Vzorkovací čas lze nastavit pro každý kanál zvlášť (externí i interní) v osmi krocích v rozmezí 3 až 480 hodinových cyklů A/D převodníku. V případě hodinové frekvence 30 MHz je to rozmezí 0,1 µs až 16 µs. Vlastní čas převodu metodou postupné aproximace je dán použitým rozlišením a pohybuje se od 6 hodinových cyklů pro rozlišení 6 bitů po 12 cyklů pro rozlišení 12 bitů.

Výrobce uvádí, že celková chyba převodu při frekvenci 30 MHz je typicky ±2 LSB a celková maximální chyba je ±5 LSB [12].

3.4 Vestavěné D/A převodníky

V mikrokontroléru STM32F407VGT6 jsou vestavěny také dva 12-bitové D/A převodníky s napěťovým výstupem. Převodníky mohou pracovat samostatně nebo synchronně. Rozlišení je nastavitelné na 8 nebo 12 bitů. Při použití rozlišení 12 bitů mohou být data pro převod zarovnána vlevo nebo vpravo.

Na výstupu převodníků lze použít vestavěný výstupní oddělovací zesilovač, který snižuje výstupní impedanci a umožňuje přímé buzení malé zátěže přímo, bez použití přídavného zesilovače.

Převod může být spuštěn softwarově, externím spouštěcím vstupem nebo interně časovačem. K přenosu dat lze také využít mechanismus DMA. Je k dispozici také lineární posuvný registr pro generování pseudonáhodného signálu. Dále je možné generovat trojúhelníkový signál.

D/A převodníky lze použít ve dvoukanálovém režimu, kdy jsou jedním zápisem do registrů ovládány oba převodníky. Lze nastavit nezávislé nebo synchronní spouštění.

3.4.1 Parametry D/A převodníků

Výrobce uvádí výstupní impedanci 15 kΩ při vypnutém oddělovacím zesilovači. Minimální zatěžovací odpor při rezistivní zátěži uvádí 1,5 MΩ pro dosažení přesnosti 1%. Při zapnutém oddělovacím zesilovači je minimální zatěžovací odpor 5 kΩ a maximální kapacita 50 pF [12].

Podle katalogového listu [12] je diferenciální nelinearita maximálně ±2 LSB. Maximální integrální nelinearita (odchylka od lineárního průběhu výstupu) je ±4 LSB. Chyba způsobená posunem výstupu (*offset*) je ±12 LSB při vnitřním referenčním napětí 3,6 V. Čas potřebný k ustálení výstupu je maximálně 6µs. Maximální frekvence změn výstupu je 1 Msps za podmínky malých změn (změna o 1 LSB).

3.5 Vývojová deska STM32F4-Discovery

STM32F4-Discovery je levná vývojová deska od firmy ST Microelectronics, jejíž hlavní součástí je mikrokontrolér STM32F407VGT6, popisovaný v této kapitole.

Vývojovou desku lze pomocí USB připojit k PC se systémem Windows XP a vyšší, přičemž přes rozhraní USB je zajištěno také napájení desky. V základním stavu je tedy k využívání desky třeba pouze PC s odpovídajícím softwarovým vybavením. Deska pak také poskytuje napětí 5 V a 3 V pro externí aplikace s proudovým odběrem maximálně 100 mA. Samozřejmě je možné desku napájet i z externího zdroje 5 V. Deska se připojuje k PC kabelem USB A – mini-B.

Součástí desky je také programovací a ladicí rozhraní ST-LINK/V2, jehož činnost a komunikace s PC je zajištěna prostřednictvím druhého mikrokontroléru (STM32F103). Rozhraní ST-LINK/V2 slouží k programování a ladění mikrokontrolérů řad STM32 a STM8 a lze jej použít jak k programování a ladění pro použitý mikrokontrolér STM32F407VGT6, tak jako samostatný programátor pro jiné aplikace. K tomuto účelu je na desce zvlášť vyvedený programovací konektor SWD (*serial wire debugging*) a dvojice přepínacích propojek (*jumper*), pomocí kterých lze snadno zvolit požadovanou funkci [14].

Všech 100 vývodů mikrokontroléru STM32F407VGT6 je vyvedeno na konektory po stranách desky. Dále je k dispozici měřicí místo (JP1) pro měření spotřeby mikrokontroléru. Na desce je také několik pájecích můstků pro změnu některých funkcí

a USB Micro-AB konektor s indikačními LED diodami pro připojení k USB OTG rozhraní mikrokontroléru STM32F407VGT6.

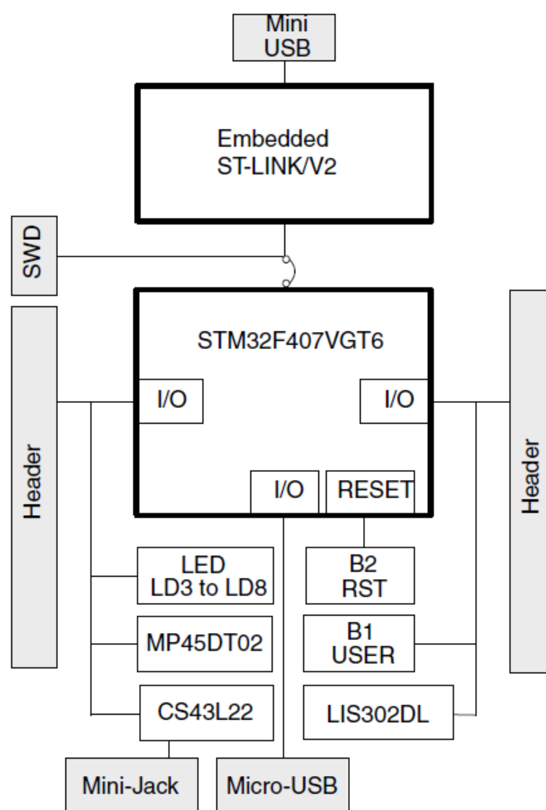
Mezi další vybavení vývojové desky STM32F4-Discovery patří:

1) nízkopříkonový tříosý lineární MEMS akcelerometr LIS302DL firmy ST Microelectronics s citlivostí ±2/±8 g [15], připojený k mikrokontroléru přes rozhraní SPI.
2) všesměrový MEMS mikrofon MP45DT02 firmy ST Microelectronics s citlivostí -26 dB a maximálním rozlišitelným akustickým tlakem 120 dBSPL [16].
3) nízkopříkonový audio D/A převodník CS43L22 firmy Cirrus Logic s integrovaným zesilovačem třídy D a sluchátkovým zesilovačem [17], připojený k mikrokontroléru přes rozhraní I^2C
4) stavové a uživatelské LED diody
5) uživatelské tlačítko a tlačítko reset

Vývojovou desku STM32F4-Discovery dodává např. internetový obchod Farnell (www.farnell.com). Deska je dodávána s nahraným demonstračním programem ve FLASH paměti mikrokontroléru. Je možné dodat také příslušenství (rozšiřující deska s konektory pro některá komunikační rozhraní, 3,5“ LCD modul, CMOS kamera).

Na Obr. 3-3 je blokové schéma zapojení desky.

Obr. 3-3: Blokové schéma zapojení vývojové desky STM32F4-Discovery [14]



4 VÝBĚR OPERAČNÍCH SYSTÉMŮ REÁLNÉHO ČASU

4.1 Operační systémy reálného času

Operační systém reálného času (*Real-time operating system*, RTOS) je víceúlohový (*multi-tasking*) operační systém pro použití v oblastech, kde je zapotřebí vykonávání určitých složitějších úloh v definovaných časových intervalech (*deadline*). To je často vyžadováno zejména v regulační a měřicí technice, robotice a obecně ve vestavěných systémech. V případě složitějších problémů již nemusí být možné při použití smyčkového řízení zaručit požadované časové vlastnosti. Použitím operačního systému reálného času lze zaručit deterministické chování i při realizaci složitějších úloh. Z toho plyne základní požadavek na operační systém reálného času, tj. deterministické chování.

Chování systému může být řízeno událostmi nebo časem, v praxi jsou často na operační systém reálného času kladeny tyto dva požadavky současně. Operační systém reálného času je tedy takový operační systém, který je schopen provádět výpočty a reagovat na události v definovaných časových intervalech (*deadlines*) [18]. Systémy pracující v reálném čase lze klasifikovat na následující skupiny.

***Soft real-time* systémy**

U tohoto typu systémů není nutné časové intervaly dodržet přesně a jejich nedodržení obvykle nevede k vážným následkům. Postačí, pokud požadavky budou splněny v určitém časovém intervalu. V [19] je uvedeno, že *soft real-time* systém je takový systém, kde naprogramovaná reakce na určitý podnět je téměř vždy dokončena ve známém konečném čase.

***Hard real-time* systémy**

Tyto systémy musí bezpodmínečně zajistit dodržení časových požadavků. Nesplnění těchto požadavků by vedlo k selhání systému a potenciálně nebezpečným situacím. Podle [19] je *hard real-time* systém takový systém, kde je garantováno, že reakce systému na podnět bude dokončena ve známém konečném čase.

Toto dělení lze pak aplikovat nejen na celé systémy, ale např. i na jednotlivé procesy v rámci jednoho systému. V praxi se pak lze často setkat s oběma uvedenými požadavky na časové chování současně a to zejména u velkých systémů [18]. Je také samozřejmě možné, aby součástí systému byly kromě *hard* a *soft real-time* procesů také procesy bez zvláštních časových požadavků [19].

Je nutné si uvědomit, že časové vlastnosti systému, tj. příslušnost k některé z uvedených skupin, stejně tak jako celková funkčnost a spolehlivost systému, nejsou

určeny prostým použitím operačního systému reálného času, ale především správným a pečlivým návrhem systému jako celku, tj. včetně všech funkčních bloků a způsobu jejich vzájemné interakce. Pokud je dodržen správný návrh a implementace systému, lze s použitím OS reálného času dosáhnout velmi spolehlivých výsledků.

Na druhé straně s sebou použití operačního systému přináší také celkové snížení výkonnosti systému a při nevhodném návrhu i problémy s konzistencí prováděných operací. Pokud je např. jedna úloha v průběhu čtení dat na určitou dobu přerušena z důvodu spuštění jiné úlohy s vyšší prioritou, je možné, že čtená data budou během této doby změněna. To bude pravděpodobně mít nežádoucí následky. Z tohoto důvodu disponují operační systémy řadou synchronizačních prostředků, které slouží nejen k synchronizaci a komunikaci mezi úlohami, ale často i k ochraně různých sdílených zdrojů.

Operační systém reálného času by měl disponovat preemptivním plánovačem úloh, který zajistí, že bude v každém okamžiku prováděna úloha s nejvyšší prioritou, která je schopna běhu. Nepreemptivní, tj. kooperativní OS lze také použít. Tyto systémy jsou jednodušší, avšak zde je nutný kvalitní návrh tak, aby každá úloha umožnila běh také úlohám ostatním a nedošlo k zablokování.

4.2 Výběr RTOS s podporou jádra ARM Cortex-M4F

Jedním z úkolů této práce je vyhledání operačních systémů reálného času, které lze provozovat na mikrokontrolérech s jádrem ARM Cortex-M4F. Jádro Cortex-M4 je kromě absence jednotky FPU totožné s jádrem Cortex-M4F. OS reálného času s podporou jádra Cortex-M4 by tedy prakticky jistě bylo možné použít i s jádrem Cortex-M4F, samozřejmě bez nároku na hardwarové výpočty s plovoucí řádovou čárkou. Pro realizaci regulačních *embedded* systémů je však právě tato vlastnost žádoucí a proto byl při vyhledávání RTOS kladen důraz na přímou podporu jádra Cortex-M4F, tj. včetně podpory FPU, bez nutnosti úprav. Tím byl stanoven první požadavek a podle něj bylo nalezeno několik následujících operačních systémů (jsou uvedeny v přibližném pořadí podle nalezení).

FreeRTOS [20]
NuttX [21]
µC/OS-II [22]
SMX RTOS [23]
ChibiOS/RT [24]
eCos [25]

Nalezené OS reálného času lze následně rozdělit na dvě skupiny podle toho, zda jsou volně šiřitelé, či nikoliv. Z uvedených šesti systémů jsou komerční pouze dva a to systémy µC/OS-II a SMX RTOS. Pro systém µC/OS-II je nabízena bezplatná zkušební verze na 45 dní, vzhledem k omezené době však tato nabídka nebyla využita.

Skupinu volně šiřitelných OS tvoří zbylé čtyři systémy. Jedním z úkolů zadání a současně cílů této práce je výběr dvou volně šiřitelných OS reálného času. První a ve značném předstihu před ostatními OS byl nalezen operační systém FreeRTOS. Tento systém je velmi dobře známý, profesionálně vyvíjený a ověřený, proto byl vybrán jako první OS k implementaci do zadaného mikrokontroléru STM32F407VGT6.

Dalšími nalezenými systémy byly NuttX a ChibiOS/RT. Systém NuttX nabízí podporu standardů POSIX a ANSI, kompaktnost a konfigurovatelnost. Z pohledu realizace regulačního *embedded* systému však nenabízí žádné zvláště užitečné funkce. Dokumentace, která je dostupná na internetových stránkách, je navíc jen velmi nepřehledná.

Operační systém ChibiOS/RT nabízí vysokou rychlost, což je pro realizaci regulačních systémů výhodné. Dále nabízí kompaktnost a také vrstvu abstrakce hardwaru (HAL), která obsahuje ovladače běžně používaných periferií. Dostupná dokumentace je přehledná a srozumitelná. Díky uvedeným vlastnostem byl tedy systém ChibiOS/RT vybrán jako druhý OS pro implementaci do daného mikrokontroléru.

Poslední uvedený systém eCos byl nalezen až později. Tento systém nabízí zajímavé vlastnosti, vzhledem k jeho komplexnosti a nedostatku času však nebyl použit.

Podrobnějšímu popisu vybraných systémů, tj. systémů FreeRTOS a ChibiOS/RT budou věnovány samostatné kapitoly (kap. 5 a 6). V následující části je uveden stručný popis ostatních nalezených OS reálného času. Systém SMX RTOS nebude popsán z důvodu pozdního nalezení použitelné dokumentace.

4.3 Popis některých nalezených operačních systémů

4.3.1 NuttX

NuttX je otevřený operační systém reálného času, který je určen pro malé vestavěné systémy. Tento systém vyhovuje standardům POSIX a ANSI a dále přebírá pro rozšíření funkcionality některé standardní API funkce ze systému Unix a některých dalších operačních systémů reálného času (např. VxWorks). Díky shodě se standardy lze snadno do systému NuttX přenést software, vytvořený pro jiný standardní operační systém. Důraz je kladen také na nízké paměťové nároky a velkou škálovatelnost od 8-bitových až po 32-bitové systémy.

Operační systém NuttX nabízí množství funkcí a obsahuje velké množství zdrojových souborů, kde většina z nich obsahuje pouze jedinou funkci. Tyto zdrojové

soubory se při překladu sestavují do knihoven. Objektové soubory z těchto knihoven se poté připojují k výslednému spustitelnému souboru v závislosti na tom, které části a funkce operačního systému jsou použity. Díky tomu je dosaženo velmi dobré modularity a škálovatelnosti a výsledný binární soubor má minimální velikost, protože obsahuje pouze funkce, které jsou systémem využívány [26].

Systém NuttX nabízí kromě úloh (*task*) také podporu vláken podle standardu POSIX (*pthread*). Tato vlákna mohou být vytvořena úlohou a následně s touto úlohou sdílí její zdroje. Úlohy jsou tedy navzájem nezávislé, zatímco vlákna vždy sdílí prostředky úlohy, která je vytvořila. Některé funkce operačního systému jsou implementovány jako interní úlohy/vlákna.

Úlohy jsou spouštěny podle priority, tj. běží vždy úloha s nejvyšší prioritou. Je také možné, aby mělo více úloh stejnou prioritu. Tyto úlohy jsou pak ve výchozím nastavení spouštěny podle algoritmu FIFO. Tento plánovací algoritmus, někdy také označovaný FCFS (*First Come, First Served*), spouští úlohy jednoduše podle pořadí příchodu jejich požadavku na běh [27]. Systém NuttX také volitelně podporuje plánovací algoritmus Round-Robin s nastavitelným časovým kvantem.

Pro synchronizaci a komunikaci mezi úlohami systém NuttX nabízí fronty zpráv, čítací semafory, *watchdog* časovače a signály. Semafory jsou základním synchronizačním prostředkem v systému NuttX a jsou preferovány pro ochranu přístupu ke zdrojům. Většina dalších synchronizačních prostředků využívá funkcionality semaforů. Semafory podporují také mechanismus dědění priorit. Jeho využití je konfigurovatelné.

Signály slouží pro asynchronní komunikaci mezi úlohami, popř. mezi úlohami a obslužnými rutinami přerušení. Úloha, mlže vyslat signál jiné konkrétní úloze. Tato úloha, jakmile je spuštěna, provede akci, která je pro tento signál uživatelsky definována.

Systém NuttX volitelně nabízí také škálovatelný jednoduchý souborový systém s podporou FAT12/16/32. Dále je k dispozici velké množství ovladačů pro různá zařízení včetně ovladače pro síťová rozhraní, USB rozhraní, sériová komunikace, rozhraní I2C, I2S, NAND paměti, ovladače pro paměťové karty MMC, SD, SDH komunikující přes rozhraní SPI nebo SDIO, ovladače pro A/D a D/A převodníky, časovače apod. Dále jsou nabízeny ovladače pro segmentové i grafické LCD displeje přes paralelní rozhraní i rozhraní SPI. Pro síťovou komunikaci je přítomna podpora protokolů TCP/IP, UDP, ICMP.

Systém také plně integruje standardní knihovnu C a matematickou knihovnu pro podporu výpočtů v plovoucí řádové čárce [21].

Kompletní seznam podporovaných architektur, rozdělených podle jádra, popř. podle výrobce, lze nalézt v [28].

4.3.2 µC/OS-II

Systém µC/OS-II je přenosný a škálovatelný operační systém reálného času, určený pro mikrokontroléry a digitální signálové procesory (DSP). Nabízí jednoduché použití a jeho zdrojový kód je vytvořen v jazyku C podle standardu ANSI C. Tento systém je implementován také ve velkém množství aplikací s vysokou funkční bezpečností.

Jádro systému je preemptivní a aplikace může využít 254 úloh, kde jedné priority může nabývat pouze jedna úloha. Pro synchronizaci lze využít semafory, příznaky událostí, mutexy s eliminací inverze priorit, fronty a zprávy typu *mailbox*. Dále jsou nabízeny časovače. Je možná alokace paměti po blocích o pevné velikosti [22].

Jsou podporovány jednotky FPU, MPU i MMU a také víceprocesorové systémy. Paměťová náročnost systému je 6 až 24 kB programové paměti v závislosti na konfiguraci. Spotřeba paměti RAM je od 500 B výše. Lze využít vestavěné měření výkonu, např. využití CPU a zásobníků nebo počítadlo přepnutí kontextu. Systém µC/OS-II je certifikován také např. podle normy IEC61508 [29].

V současné době je podporováno přibližně 50 procesorových architektur od různých výrobců [30].

K systému µC/OS-II je dostupná také rozsáhlá dokumentace ve formě knihy, kterou lze stáhnout z [31].

4.3.3 eCos

Operační systém eCos je volně šiřitelný a otevřený operační systém reálného času, který je určen pro *embedded* aplikace. Jeho název je zkratkou slovního spojení *Embedded Configurable Operating System*, což naznačuje velmi dobrou konfigurovatelnost systému.

Jednou z hlavních vlastností operačního systému eCos je konfigurační systém. Tento systém dovoluje při tvorbě aplikací ovlivňovat jak funkcionalitu jednotlivých částí operačního systému, tak i jejich implementaci, což obvykle není možné. To v podstatě umožňuje vytvořit speciální operační systém, který přesně splňuje požadavky konkrétní aplikace. Uvedená vlastnost činí z operačního systému eCos velmi univerzální systém, použitelný v širokém rozsahu *embedded* aplikací. Je také zajištěna minimalizace paměťových nároků, veškerá nepotřebná funkcionalita systému může být odstraněna. S konfiguračním systémem souvisí také architektura operačního systému eCos. Ten se skládá z různých komponent, což poskytuje standardní mechanismus pro rozšiřování funkcionality systému. Jednotlivé komponenty jsou buď přímou součástí distribuce nebo je lze získat od komunity vývojářů, popř. také jako komerční produkty třetích stran [32]. Systém eCos se skládá z následujících komponent [33].

Vrstva abstrakce hardwaru (HAL)

Tato vrstva slouží k oddělení implementace závislé na dané architektuře od ostatních částí systému, které ji využívají a pro které poskytuje jednotné aplikační rozhraní (API). Celá vrstva HAL je implementována v jazyku C a jazyku symbolických adres. Rozhraní je implementováno pomocí maker, což dovoluje jeho efektivní využití, protože je eliminováno volání funkcí. Nevýhodou tohoto přístupu je pak zvyšování paměťových nároků. Je kladen důraz na snadnou rozšiřitelnost pro nové platformy. Vrstva abstrakce hardwaru se skládá ze tří modulů.

První modul definuje architekturu. V systému eCos je za samostatnou architekturu považována každá rodina procesorů. Tento modul obsahuje kód pro inicializaci procesoru, správu přerušení a přepínání kontextu a některé další funkce, specifické pro určitou rodinu procesorů.

Druhý modul definuje variantu, tj. konkrétní procesor dané rodiny. Kód obsažený v tomto modulu může obsahovat ovladače pro různá zařízení, kterými konkrétní procesor/mikrokontrolér disponuje.

Třetí modul definuje platformu. Platformou je zde myšlen určitý hardware, např. vývojová deska, obsahující určitý mikrokontrolér. Modul obsahuje inicializaci hardwaru, konfiguraci mikrokontroléru apod.

Jádro systému (*Kernel*)

Jádro představuje základní část systému a poskytuje standardní funkce, požadované od OS reálného času, tj. zpracování přerušení, plánování úloh, synchronizace. Tyto jednotlivé části jsou konfigurovatelné a umožňují tedy přizpůsobení systému požadavkům aplikace.

Jádro systému je implementováno v jazyku C++. Oficiální aplikační rozhraní pro jazyk C++ však neexistuje. Jádro také podporuje rozhraní API podle standardů µITRON [34] a POSIX.

Operační systém eCos nabízí dvě různé možnosti plánování úloh, přičemž použita může být jen jedna.

Prvním plánovacím mechanismem je *multilevel queue scheduler*. Ten dovoluje provádění několika úloh se stejnou prioritou. Počet úrovní priorit je konfigurovatelný od 1 do 32 s tím, že nejvyšší priorita odpovídá hodnotě 0 a nejnižší priorita hodnotě 31. Mezi různými úrovněmi plánovač rozhoduje podle priorit, tj. je spuštěna úloha s nejvyšší prioritou.. Úlohy o stejné prioritě jsou spouštěny po časových úsecích (*timeslicing*), tj. v postatě algoritmus Round-Robin.

Druhý plánovací mechanismus je *bitmap scheduler*. Tento plánovač umožňuje rovněž více priorit, avšak není možné, aby existovalo více úloh se stejnou prioritou. To zjednodušuje plánování a tento plánovač je tak velmi efektivní. Využitelné úrovně priorit jsou stejné jako u prvního plánovacího mechanismu.

Volba plánovacího mechanismu závisí na potřebách konkrétní aplikace. Podrobnější popis plánování lze nalézt v [33].

Pro synchronizaci nabízí systém eCos mutexy, semafory, podmíněné proměnné, příznaky, zprávy a zámky *spinlock* pro víceprocesorové systémy typu SMP [35]. Popis synchronizačních mechanismů lze nalézt v [36] a [33].

Jádro systému dále nabízí čítače, časovače a alarmy, alokaci paměti, využívající bloky pevné velikosti (*memory pools*), dále prostředky pro ladění a trasování.

Knihovna ISO C a matematická knihovna

Systém eCos je kompatibilní se standardní knihovou ISO C a dále nabízí matematickou knihovnu, která může pracovat ve čtyřech různých módech kompatibility (výchozí je kompatibilita se standardem POSIX) [33].

Ovladače zařízení

Ovladače pro sériovou komunikaci, ethernet, SPI, I^2C, CAN, A/D převodníky, *framebuffer* pro zobrazování, *watchdog* obvody. Lze také vytvořit vlastní ovladače. Ovladače mohou být vrstvené, tj. složitější ovladače mohou využívat jiných ovladačů. Aplikace s ovladačem zařízení komunikuje prostřednictvím standardního vstupně/výstupního rozhraní.

Podpora programu GDB (GNU *debugger*)

Pro ladění systém nabízí software pro komunikaci aplikace s ladicím programem GNU *debugger*.

Systém eCos zahrnuje kromě vlastní funkcionality také všechny nástroje, potřebné k vývoji vestavěných aplikací. Jedná se o konfigurační nástroje, dále GNU překladače, linkery, ladicí prostředky a simulátory. V současné době podporuje systém eCos 18 procesorových architektur [32].

5 OPERAČNÍ SYSTÉM FREERTOS

FreeRTOS je volně šiřitelný, otevřený operační systém reálného času, určený pro použití v menších vestavěných (*embedded*) zařízeních.

Projekt FreeRTOS vznikl s cílem poskytnout kvalitní, volně šiřitelný operační systém reálného času, který by byl jednoduše použitelný. Zakladatelem projektu je Richard Barry, ředitel společnosti Real Time Engineers Ltd., která v současné době tento systém vyvíjí a udržuje. Hlavními cíli vývoje systému jsou snadná použitelnost, robustnost a nízká paměťová náročnost [37]. Podle [38] je pro běh jádra na mikrokontroléru s jádrem ARM7 při plné optimalizaci třeba 236 bytů paměti RAM a navíc 64 bytů pro každou vytvořenou úlohu. Při této konfiguraci je velikost jádra v rozmezí 5 až 10 kB.

Základní funkcionalita operačního systému FreeRTOS představuje pouze vlastní plánovač (*scheduler*) a správu úloh, komunikaci a synchronizaci mezi úlohami a časování. Systém lze dále rozšířit o přídavné funkce, např. síťové rozhraní.

V současné době je podle [39] oficiálně podporováno více než 30 různých procesorových architektur. Plánování úloh může být preemptivní, kooperativní, popř. hybridní. Pro synchronizaci a komunikaci mezi úlohami lze použít fronty, semafory, mutexy a rekurzivní mutexy. Dále systém nabízí efektivní softwarové časovače [40] a příznaky událostí [41]. Lze využít makra pro sledování běhu aplikace [42] a detekci přetečení zásobníku [43].

Počet úloh a možných priorit není softwarově omezen, je možné přiřadit více úlohám stejnou prioritu. Také je možnost podpory jednotky správy a ochrany paměti (MPU) pro mikrokontroléry s jádry ARM Cortex-M3.

FreeRTOS je velmi populární a rozšířený operační systém pro *embedded* zařízení [44]. Je šířen pod upravenou licencí GNU GPL. Tato úprava spočívá v umožnění použití systému v komerčních aplikacích a dále osvobozuje od povinnosti zveřejnit kód aplikace, pokud nedošlo k úpravě jádra systému, např. ke změně nebo rozšíření jeho funkčnosti. Je také nabízena komerční licence s profesionální podporou a zárukami. Více informací o licencích lze najít v [45]. Z operačního systému FreeRTOS je také odvozen komerční systém SAFERTOS [46], který splňuje normu funkční bezpečnosti IEC 61508 SIL 3 [47].

5.1 Podporované architektury a nástroje

Jak bylo uvedeno, operační systém FreeRTOS oficiálně podporuje více než 30 architektur mikrokontrolérů. V [48] je uveden oficiální seznam výrobců s podporovanými rodinami mikrokontrolérů a vývojovými nástroji.

Od výrobce ST Microelectronics je podporována rodina mikrokontrolérů STM32 s jádry ARM Cortex-M0, Cortex-M3 a Cortex-M4F, dále rodina STR7 založená na

jádru ARM7 a rodina STR9 s jádrem ARM9. Součástí systému je mimo jiné také demo aplikace přímo pro mikrokontrolér STM32F407. Podporované vývojové nástroje pro mikrokontroléry tohoto výrobce jsou IAR, Atollic TrueStudio, GCC, Keil, Rowley CrossWorks.

Mezi další oficiálně podporované výrobce patří:

Altera
Atmel
Cortus
Cypress
Energy Micro
Freescale
Fujitsu
Infineon
Luminary Micro
Microchip
NEC
Microsemi (Actel)
NXP
Renesas
Silicon Labs
Texas Instruments
Xilinx

Je podporována i architektura x86 a je dostupný také simulátor systému FreeRTOS pro operační systém Windows [49]. Dále jsou dostupné i uživatelské modifikace pro několik dalších rodin procesorů, které však nejsou oficiálně podporovány. Detailní seznam výrobců a podporovaných rodin mikrokontrolérů včetně popisu demo aplikací lze nalézt v [50].

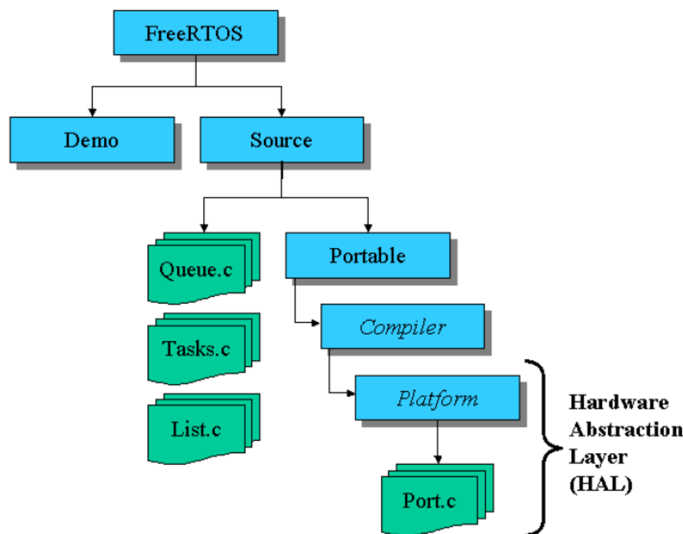
5.2 Struktura zdrojových souborů

Operační systém FreeRTOS je distribuován ve formě zdrojových souborů. Součástí distribuce je kromě jádra systému a jeho volitelných součástí také velké množství demonstračních aplikací pro podporované architektury a překladače, popř. pro různá integrovaná vývojová prostředí (IDE).

Distribuci lze zdarma stáhnout ve formě standardního zip archivu (.zip) nebo ve formě samorozbalovacího archivu (.exe) [51]. V době psaní této práce je aktuální verze 8.0.0. Ve staženém hlavním adresáři (**FreeRTOSV8.0.0**) se nachází kromě textového souboru s popisem a odkazů na internetové stránky projektu FreeRTOS také dva podadresáře. Podadresář **FreeRTOS** obsahuje vlastní zdrojové soubory operačního

systému, demonstrační aplikace a licenční informace. Podadresář **FreeRTOS-Plus** pak obsahuje některé přídavné funkce pro systém FreeRTOS a produkty třetích stran, např. vstupně/výstupní funkce, souborový systém atd. [52]. Další popis bude věnován struktuře adresáře **FreeRTOS**. Ta je znázorněna na Obr. 5-1.

Obr. 5-1: Struktura distribuce operačního systému FreeRTOS [55]



V podadresáři **Source** se nachází zdrojové soubory jádra systému včetně volitelných součástí. Vlastní jádro systému je reprezentováno pouze třemi zdrojovými soubory (**tasks.c**, **queue.c** a **list.c**) [53]. Tyto soubory představují funkcionalitu pro správu úloh, front a seznamů. Dále se zde nachází zdrojové soubory pro volitelné součásti systému. Jsou to **timers.c** pro funkce softwarových časovačů, **croutine.c** pro využití koprogramů a **event_groups.c** pro příznaky událostí.

Adresář **Source** dále obsahuje podadresáře **include** a **portable**. V podadresáři **include** jsou umístěny všechny potřebné hlavičkové soubory. Podadresář **portable** pak obsahuje různé implementace správy paměti (podadresář **MemMang**) a dále zdrojové a hlavičkové soubory, které jsou specifické pro konkrétní překladač a architekturu. Struktura těchto souborů je následující.

Nejprve je provedeno třídění podle překladače (popř. použitého IDE). Každý překladač je reprezentován vlastním adresářem. V tomto adresáři se pak nachází množství dalších podadresářů, kde každý představuje jednu z podporovaných architektur. Každý z těchto podadresářů pak obsahuje hlavičkový soubor **portmacro.h**, definující např. použité datové typy, a zdrojový soubor **port.c**, který definuje další specifika dané architektury a jeho velká část je psána v jazyce symbolických adres. Tyto dva soubory tedy v podstatě tvoří hardwarovou abstrakci (HAL). Podadresáře některých architektur obsahují další soubory, které jsou součástí HAL.

Chceme-li např. použít mikrokontrolér STM32F407VGT6 (jádro ARM Cortex-M4F) a překladač GCC, pak soubory vrstvy HAL nalezneme v adresáři

FreeRTOSV8.0.0/FreeRTOS/Source/portable/GCC/ARM_CM4F. Tyto soubory, stejně jako některou z implementací správy paměti (podadresář **MemMang**), je pak samozřejmě třeba přidat ke zdrojovým souborům aplikace.

V adresáři **FreeRTOSV8.0.0/FreeRTOS/Demo** se nachází demonstrační aplikace pro různé architektury. Tyto aplikace lze využít jako vzor k tvorbě aplikace vlastní. Součástí demo aplikace je i konfigurační soubor **FreeRTOSConfig.h**, který slouží k uživatelskému nastavení systému, např. frekvence systémových hodin, maximální velikost haldy pro alokaci paměti, použití mutexů apod. Více informací o tomto konfiguračním souboru lze nalézt v [54].

5.3 Správa úloh

Základní a nejdůležitější funkcí operačního systému je správa úloh. Jednotlivé úlohy jsou spravovány pomocí řídicích bloků úloh (Task Control Block, dále jen TCB). Tento blok obsahuje informace, které jsou využívány jádrem operačního systému. Každá vytvořená úloha má vlastní TCB, který plně určuje její stav.

V [55] je uveden popis TCB pro FreeRTOS verze 4.1.3 z konce roku 2006 (dostupná z [56]), v [57] je pak uvedena struktura TCB pro blíže neurčenou verzi FreeRTOS, pravděpodobně z roku 2011 (dle data vytvoření souboru). Z porovnání těchto dvou struktur je vidět, že TCB se měnil s spolu s vývojem systému. Byla přidána např. nastavení použití jednotky ochrany a správy paměti (MPU) pro mikrokontroléry, které touto jednotkou disponují, nebo použití mutexů (funkcionalita pro mutexy byla uvedena ve verzi 4.5.0 v roce 2007 [58]). V Tab. 5-1 je uvedena struktura TCB použitá ve verzi FreeRTOS 8.0.0. Informace byly převzaty přímo ze zdrojového souboru **tasks.c**, který je součástí jádra systému. TCB je zde definován jako typ struktury, obsahující potřebné proměnné různých datových typů. Většina těchto proměnných je definována podmíněně, v závislosti na uživatelské konfiguraci systému a použitém mikrokontroléru. Úloha může nabývat jednoho ze čtyř stavů [59]. Na Obr. 5-2 je diagram přechodů mezi stavy úloh, převzatý z [60]. Tyto stavy jsou následující.

Běžící (*Running*)

Stav, ve kterém se nachází aktuálně běžící úloha. V tomto stavu může být v jednom okamžiku pouze jedna úloha a tato úloha má přístup k procesoru. Obvykle je to úloha s nejvyšší prioritou. Stav Běžící může úloha opustit buď sama, nebo rozhodnutím plánovače (*scheduler*), který také rozhoduje o spouštění úloh.

Připravená/čekající (*Ready*)

V tomto stavu se může nacházet více úloh, které mají k dispozici všechny požadované zdroje a pouze čekají na spuštění. Čekající úlohy jsou seřazeny v seznamu podle priority a jsou postupně spouštěny, tj. přesouvány do stavu Běžící.

Blokovaná (*Blocked*)

V tomto stavu se nachází úlohy, které čekají na nějakou událost nebo zdroj. Událost může být časová (např. při cyklickém spouštění úlohy) nebo externí. Zdrojem mohou být fronty a semafory. Úloha se ve stavu Blokovaná může nacházet pouze po určitou dobu, po jejímž uplynutí je odblokována a poté přesunuta do stavu Připravená

Pozastavená (*Suspended*)

Do tohoto, popř. z tohoto stavu se úloha může dostat pouze zavoláním příslušné funkce aplikačního rozhraní (API) operačního systému. Pro stav Pozastavená není možné určit časový limit.

Tab. 5-1: Struktura TCB v systému FreeRTOS

název proměnné	význam
`pxTopOfStack`	ukazatel aktuálního vrcholu zásobníku úlohy
`xMPUSettings`	nastavení pro jednotku MPU
`xGenericListItem`	stav úlohy [1]
`xEventListItem`	místo v seznamu událostí
`uxPriority`	priorita úlohy
`pxStack`	ukazatel začátku zásobníku
`pcTaskName`	název úlohy pro ladění
`pxEndOfStack`	ukazatel konce zásobníku
`uxCriticalNesting`	hloubka vnoření pro kritickou sekci
`uxTCBNumber`	počet vytvoření TCB (použití pro ladění)
`uxTaskNumber`	číslo úlohy (pro trasování)
`uxBasePriority`	poslední priorita přidělená úloze (dědění priorit)
`pxTaskTag`	tag hodnota přidělená úloze [2]
`ulRunTimeCounter`	čas běhu úlohy (úloha ve stavu Běžící)
`xNewLib_reent`	reentrantní struktura pro knihovnu Newlib [3]

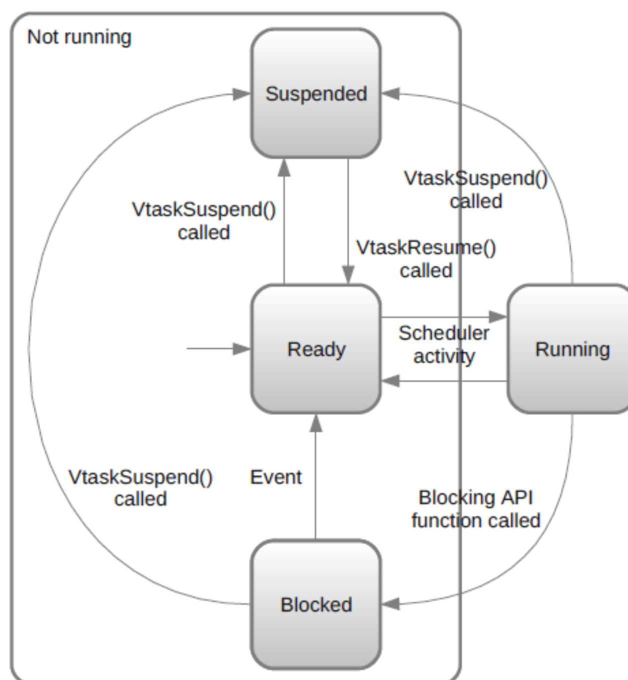
---

[1] Jedná se o položku seznamu, která reprezentuje danou úlohu a náleží do některého ze seznamů připravených, blokovaných nebo pozastavených úloh. Tím je v podstatě určen stav úlohy.

[2] Tuto hodnotu jádro systému nepoužívá a lze ji využít libovolně v aplikaci. Může představovat libovolnou hodnotu včetně ukazatele na funkci. V [61] a [62] lze nalézt popis API funkcí pro nastavení a čtení této hodnoty včetně příkladů použití.

[3] Newlib je knihovna pro jazyk C a je určena pro použití ve vestavěných systémech. [63] Ukazatel na tuto strukturu je použit pro předávání kontextových informací při použití knihovních funkcí. Více např. v [64] a [65]. Podpora knihovny Newlib v systému FreeRTOS je volitelná.

Obr. 5-2: **Přechodový diagram stavů úloh v systému FreeRTOS [60]**



5.4 Plánování a priority

Jak bylo výše uvedeno, plánovač (*scheduler*) operačního systému FreeRTOS může být preemptivní nebo kooperativní. Při každém uplynutí periody systémových hodin je vyvoláno přerušení a plánovač pracuje jako obslužná rutina tohoto přerušení.

Podle [55] je v této obslužné rutině vždy nejprve resetován časovač, měřící systémový čas. Další akce jsou závislé na nastaveném režimu.

V preemptivním režimu je pak uložen kontext právě běžící úlohy. Následně je inkrementován systémový čas. Tato událost může způsobit odblokování některých úloh, proto je testováno i toto. Pokud dojde k odblokování úlohy s vyšší prioritou, něž má úloha právě běžící, dochází k přepnutí kontextu (*context switch*). Plánovací mechanizmus rozhoduje o spouštění úloh pouze na základě priority. Každé úloze je přidělena priorita v rozsahu od 0 (úlohy s nejnižší prioritou) do maximální hodnoty. Tuto maximální prioritu lze nastavit uživatelsky podle požadavků aplikace. Nakonec je kontext obnoven a obsluha přerušení končí.

Pokud je nastaven kooperativní režim, dojde pouze k inkrementaci systémového času.

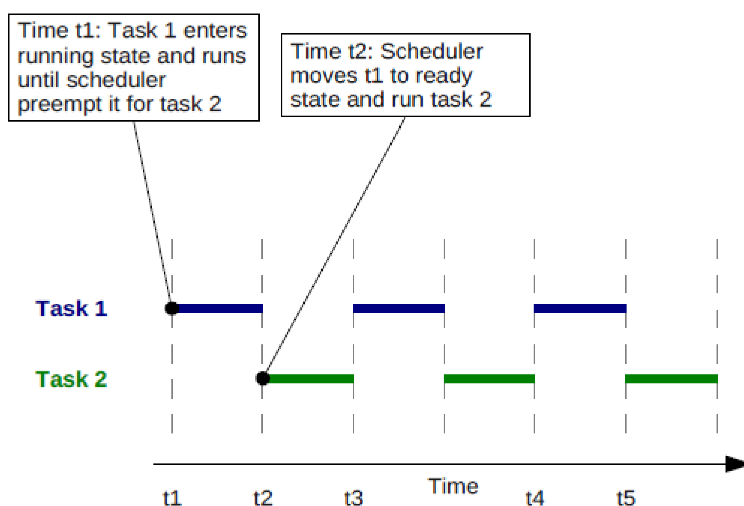
Pokud se ve stavu Připravená nachází více úloh se stejnou prioritou a není tedy možné rozhodnout, která úloha by měla běžet, je pro plánování aplikován algoritmus Round-Robin [66]. Tento algoritmus spočívá v přidělení určitého časového kvanta (*quantum, time-slice*), po který je daná úloha vykonávána. Po uplynutí tohoto času je

úloha přerušena a je spuštěna úloha další. Takto je každé úloze přidělen stejný čas pro běh. Je samozřejmě možné provést přepnutí úloh dříve než uplyne vymezené časové kvantum, pokud např. právě běžící úloha sama vyšle požadavek k přepnutí nebo dojde-li k odblokování úlohy s vyšší prioritou.

Volba časového kvanta je kompromisem mezi dobou čekání jednotlivých úloh a frekvencí přepínání kontextu. V [66] je uvedeno, že pro deset úloh (procesů) je při časovém kvantu 100 ms a době přepnutí kontextu 5 ms doba čekání poslední úlohy 945 ms a procesorový čas spotřebovaný pro přepínání kontextu představuje 4 % z celkového času. Oproti tomu při časovém kvantu 10 ms je doba čekání poslední úlohy pouze 135 ms, ale čas spotřebovaný na přepínání kontextu se zvýšil na více než 33 %.

V systému FreeRTOS je časové kvantum, přidělené jednotlivým úlohám, rovno periodě systémových hodin. Při každém přerušení po uplynutí periody hodin je provedeno přepnutí na další úlohu [60]. Na Obr. 5-3 je příklad spouštění dvou úloh se stejnou prioritou.

Obr. 5-3: Znázornění spouštění úloh pomocí algoritmu Round-Robin [60]



Operační systém FreeRTOS dále nabízí využití kritických sekcí. Kritická sekce je úsek kódu, který z nějakého důvodu nemá být přerušován. Pro tento účel lze použít definovaná makra pro vstoupení do kritické sekce a její opuštění. Při vstupu do kritické sekce jsou zakázána přerušení a při opuštění této sekce jsou přerušení opět povolena.

Kritické sekce ve FreeRTOS mohou být vnořené. Pokud funkce v kritické sekci zavolá jinou funkci, která taktéž obsahuje kritickou sekci, dochází k vnoření. Každá úloha má vlastní proměnnou pro uložení hloubky vnoření do kritické sekce (viz Tab. 5-1). Při každém vstupu do kritické sekce je tato proměnná inkrementována, při každém opuštění kritické sekce je dekrementována.

Plánovač úloh je také možné pozastavit. Na rozdíl od kritické sekce, kde jsou zakázána přerušení, je při pozastavení plánovače obsluha přerušení povolena. Pozastavení lze použít pro úlohy, které potřebují běžet např. po delší dobu a zároveň je zapotřebí zachovat aktivní obsluhu přerušení.

Podle [55] může být pozastavení plánovače vnořené, tj. lze plánovač pozastavit několikrát. Pak je třeba jeho činnost obnovit stejným počtem volání příslušné funkce. Při obnovení funkce plánovače je otestováno, zda v době, kdy byla jeho činnost pozastavena, nedošlo k odblokování některých úloh. Pokud došlo k odblokování úlohy s vyšší prioritou, než má úloha právě běžící, dojde okamžitě po obnovení činnosti k přepnutí na tuto úlohu. V době, kdy je pozastavena činnost plánovače, nesmí být volány API funkce, které by měly za následek přepnutí kontextu [67].

5.5 Synchronizace a komunikace

Pro synchronizaci a komunikaci nabízí systém několik prostředků. Základním z těchto prostředků jsou fronty. Mechanizmu front pak využívají ostatní synchronizační a komunikační prostředky operačního systému FreeRTOS [60].

5.5.1 Fronty

Jak bylo uvedeno, fronty jsou hlavním prostředkem komunikace mezi úlohami. Lze je použít ke sdílení dat mezi úlohami nebo mezi úlohami a obslužnými rutinami přerušení. Ve většině případů je fronta využívána jako paměť typu FIFO (*First In, First Out*) s tím, že data jsou posílána na konec fronty. Lze také posílat data na začátek fronty.

Fronta je charakterizována dvěma parametry. Prvním je velikost dat, tj. velikost jedné každé datové buňky, uložené ve frontě. Druhým parametrem je délka fronty, tj. počet buněk o dané velikosti. Tyto parametry jsou pevně nastaveny při vytvoření fronty.

Data jsou frontami přenášena pomocí kopií, tj. pokud úloha pošle do fronty data, jsou tato data zkopírována. Podle [68] je tento přístup vhodný mimo jiné z následujících důvodů.

Data uložená v proměnných mohou být do fronty poslána přímo, bez nutnosti vytvářet pomocné proměnné. Stejně tak je možné data z fronty ukládat přímo do proměnných. Po odeslání do fronty může úloha s proměnnou ihned pracovat, protože její obsah byl zkopírován.

Dále je možné do fronty předávat data odkazem, pokud je místo samotných dat do fronty zkopírován ukazatel na tyto data. Další možností je do fronty odeslat strukturu, která obsahuje ukazatel na data a také proměnnou, která udává velikost těchto dat. Tímto je možno dosáhnout různé délky dat v jedné frontě. Podobným přístupem lze do fronty posílat různé typy dat a zpráv, kdy způsob zpracování dat závisí na jejich typu, poslaném do fronty spolu s těmito daty.

Alokaci paměti pro fronty obstarává jádro systému a pouze jádro má k této paměti přístup. To zajistí předání dat i v případě, že komunikující úlohy mají k dispozici různé paměťové prostory (z důvodu ochrany paměti).

Pro použití v obslužných rutinách přerušení jsou určeny zvláštní funkce pro práci s frontami, což dovoluje jisté zjednodušení. Více informací lze nalézt v [68].

Pokud se úloha pokouší číst z prázdné fronty, popř. zapisovat do plné fronty, je zablokována (přesunuta do stavu Blokovaná). Při volání funkcí pro čtení nebo zápis do fronty je také možné nastavit časový limit, po který je úloha blokována.Ve stavu Blokovaná zůstává úloha až do doby, kdy jsou ve frontě k dispozici data (popř. je k dispozici volné místo pro zápis) nebo když dojde k dosažení nastaveného časového limitu pro blokování. Poté je úloha přesunuta do stavu Připravená.

Pokud je blokováno více úloh, čekajících na frontu, je jako první odblokována úloha s nejvyšší prioritou [68]. V [60] je dále uvedeno, že pokud je blokováno více úloh, čekajících na frontu, a majících stejnou prioritu, jako první bude odblokována úloha, která první odeslala požadavek.

Při čtení z fronty je přečtený prvek odebrán, je však možné použít speciální funkci pro čtení, která zajistí, aby byl přečtený prvek zachován.

5.5.2 Binární semafory

Binární semafory představují nejjednodušší způsob synchronizace úloh popř. úloh a přerušení. Binární semafor je v podstatě fronta s délkou rovnou jedné, tj. lze do něj uložit pouze jeden prvek. Tento semafor tedy může být buď prázdný nebo plný, tj. nabývá pouze dvou stavů. Odtud název binární [69].

Úlohy a obslužné rutiny přerušení, využívající binárních semaforů, pak nepracují přímo s obsahem této jednoprvkové fonty, ale pouze s informací, zda je plná nebo prázdná. Obsluha semaforu je obdobná jako u fronty. Úlohy mohou semafor nastavit (*give*) nebo přečíst (*take*). Přečtením je semafor vyprázdněn a může být opětovně nastaven. Podobně jako u front je úloha při pokusu přečíst semafor, který nebyl nastaven, přesunuta do stavu Blokovaná až do doby, kdy dojde k jeho nastavení. Opět je možné nastavit časový limit pro odblokování v případě, že do této doby není semafor nastaven.

Při nastavování semaforu k blokování nedochází. Pokud se úloha pokouší nastavit semafor, který je již nastaven, vrátí příslušná funkce chybové hlášení.

V [69] je uveden příklad použití binárního semaforu pro synchronizaci obsluhy periferie. Úloha, která tuto periferii obsluhuje, je většinu času ve stavu Blokovaná. Při potřebě obsluhy periferie je vyvoláno přerušení a obslužná rutina tohoto přerušení pouze nastaví binární semafor. Při nastavení semaforu dochází k odblokování úlohy a tato obslouží periferii.

5.5.3 Čítací semafory

Tyto semafory lze považovat opět za fronty, v tomto případě s délkou větší než jedna. Čítací semafory tedy mohou být přečteny vícekrát. Tak jako u binárních semaforů zde nezáleží na obsahu, pouze na informaci, jestli je semafor prázdný či nikoliv. Nastavování a čtení čítacích semaforů probíhá stejným způsobem jako u semaforů binárních [60].

Při vytvoření čítacího semaforu jsou určeny dva jeho parametry. První je maximální počet nastavení (*give*) semaforu, tj. v podstatě jeho délka. Druhým parametrem je počáteční nastavení. Toto počáteční nastavení udává, kolikrát je možné semafor přečíst (*take*) bez toho, aby jej bylo nutno znovu nastavit.

Typickým využitím čítacích semaforů je počítání zpracovaných událostí [70]. Při příchodu události je čítací semafor nastaven, při jejím zpracování jinou úlohou je přečten. Obsah semaforu pak udává rozdíl počtu událostí přišlých a zpracovaných.

Dalším uvedeným příkladem použití při správě zdrojů, které jsou početně omezeny. Toto omezení je nastaveno při vytvoření semaforu. Pokud úloha potřebuje zdroj využít, přečte (*take*) semafor. Pokud je obsah semaforu vyčerpán, není již k dispozici žádný zdroj. Po skončení práce se zdrojem daná úloha nastaví (*give*) semafor zpět a zdroj lze znovu použít.

5.5.4 Mutexy

Mutexy lze použít k řešení vzájemného vyloučení (*mutual exclusion*) a obvykle slouží ke hlídání přístupu k nějakému zdroji. V podstatě se jedná o binární semafory. Stejně jako u semaforů lze opět nastavit časový limit pro přečtení mutexu. Úloha, která má využívat zdroj, musí mutex přečíst (*take*). Tím získává právo využívat tento zdroj. Po použití úloha opět mutex nastaví (*give*) a přístup k hlídanému zdroji je možný [71].

Na rozdíl od binárních semaforů využívají mutexy mechanizmus dědění priorit. Tento mechanizmus má zajistit, aby úloha s vysokou prioritou byla blokována po co nejkratší dobu. Pokud je blokována úloha s vyšší prioritou, čekající na mutex, který byl již přečten (*take*) jinou úlohou s prioritou nižší, je priorita této úlohy dočasně zvýšena na hodnotu priority čekající úlohy. Tímto je podle [71] v některých případech zajištěna minimalizace vzniku inverze priorit, avšak nedochází k úplnému odstranění tohoto jevu.

Podle [60] použití mutexů zvyšuje celkovou komplexnost aplikace a pokud je to možné, je vhodné se jejich použití vyhnout.

Zvláštním typem mutexů jsou rekurzivní mutexy [72]. Tyto mutexy lze přečíst (*take*) několikrát. Pokud úloha přečte několikrát rekuzivní mutex, je nutné, aby jej stejným počtem nastavení vrátila do stavu, kdy je k dispozici k přečtení také ostatním úlohám.

5.5.5 Příznaky událostí

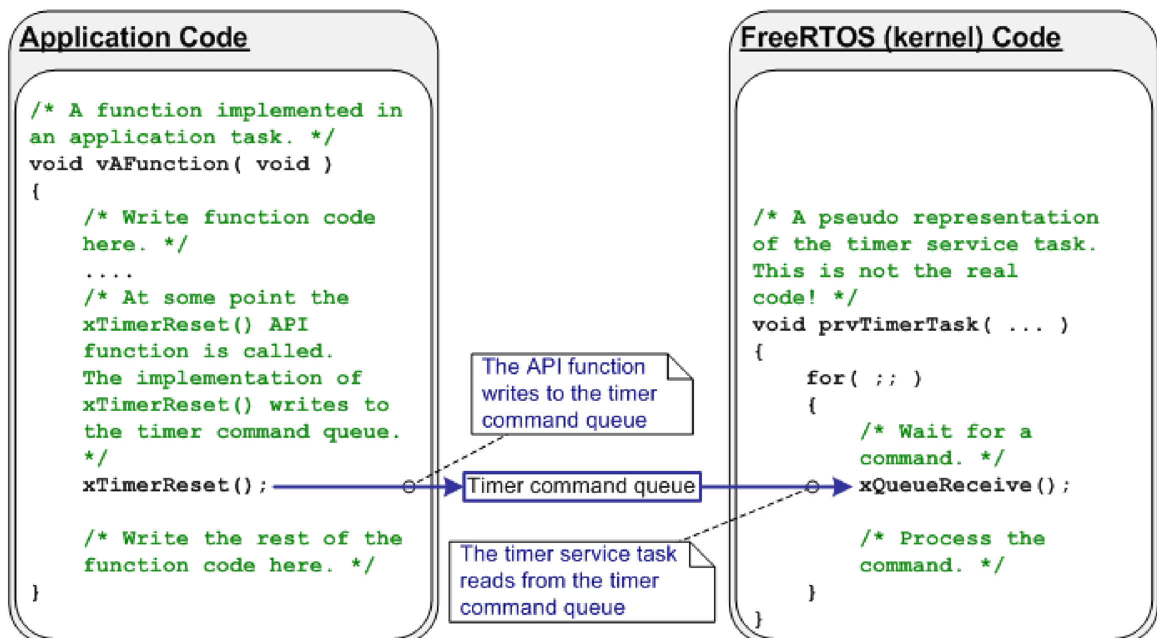
Novinkou uvedenou ve verzi FreeRTOS 8.0.0 jsou příznaky událostí (*event flag*), popř. jejich skupiny (*event group*). Tyto příznakové bity a skupiny patří k synchronizačním prostředkům a lze je použít k indikaci příchodu události nebo několika událostí. Více informací a odkazy na příklady použití lze nalézt v [41].

5.6 Časování

K časovému spouštění funkcí lze využít softwarové časovače. Před použitím musí být softwarový časovač nejprve vytvořen. Po spuštění měří časovač nastavenou dobu a po jejím uplynutí je zavolána příslušná funkce (*timer callback function*). Podle [73] nesmí taková funkce volat API funkce, které by způsobily blokování (např. pozastavení úlohy nebo nastavení nenulového časového limitu pro čtení z fronty).

Vzhledem k tomu, že softwarové časovače jsou volitelnou součástí systému, není jejich funkcionalita přímo součástí jádra systému. Místo toho je realizována samostatně v obslužné úloze časovače (*timer service task, daemon task*). Některé API funkce pro časovače, které jsou využívány v aplikaci, pak pro komunikaci s obslužnou úlohou časovače využívají standardní fronty (*timer command queue*). Tato fronta je využívána pouze systémem a nelze k ní přistupovat přímo [74]. Obr. 5-4 ukazuje vztah aplikace, komunikační fronty a obslužné úlohy časovače.

Obr. 5-4: Vzájemný vztah mezi aplikací a obsluhou softwarového časovače [74]



K dispozici jsou dva typy časovačů. Prvním typem je jednorázový časovač (*one-shot timer*), který po uplynutí nastaveného času spustí požadovanou funkci pouze jednou a dále již čas neměří. Může být však restartován aplikací. Druhým typem časovače je automatický časovač (*auto-reload timer*). Tento časovač je po uplynutí nastaveného času a spuštění příslušné funkce automaticky restartován. Tím lze dosáhnout periodického spouštění funkcí [75].

Časovač je možné za běhu resetovat. Pokud se tak stane, dochází k přepočtu měřené doby a měření času je tedy relativní k okamžiku, kdy došlo k resetu. V [76] je uveden příklad použití pro ovládání podsvětlení LCD displeje, kdy je zapotřebí podsvětlení vypnout 5 s po stisknutí posledního tlačítka.

5.7 Správa paměti

Při vytváření úloh, front, semaforů atd. je třeba vyhradit a přidělit těmto instancím potřebnou paměť RAM. Použití standardních funkcí pro alokaci paměti (**`malloc()`**, **`free()`**) však obvykle není vhodné. Tyto funkce nejsou vždy v *embedded* systémech dostupné, jsou paměťově náročné a jsou nedeterministické, tj. čas potřebný k jejich vykonání nemusí být vždy stejný [77]. Podobné nevýhody standardních funkcí jsou uvedeny také v [60], kde je navíc zmíněno i nebezpečí preempce při alokaci a dále fragmentace paměti. Z těchto důvodů je třeba spravovat paměť jinak.

Operační systém FreeRTOS má správu paměti implementovanou ve vrstvě HAL, tedy oddělenou od jádra systému. Jádro pak pouze využívá nadefinované funkce bez ohledu na implementaci. Výhodou tohoto řešení je možnost využívat různé implementace správy paměti (viz dále), popř. implementovat vlastní správu, která přesně vyhovuje konkrétní aplikaci.

V současné době jsou spolu se systémem FreeRTOS k dispozici čtyři různé implementace správy paměti Tyto implementace definují alokační funkce, které jsou využívány jádrem systému a volány některými API funkcemi. Navzájem se liší způsobem a možnostmi alokace a každá je určena pro jiný druh aplikací. Každá implementace se pak nachází v samostatném zdrojovém souboru.

Varianta 1

Tato implementace je nejjednodušší. Počítá se pouze s jednorázovou alokací paměti, tj. alokovaná paměť nemůže být uvolněna.

V [77] je uvedeno, že většina vestavěných aplikací vytváří všechny potřebné úlohy, fronty a synchronizační prostředky při startu, ještě před spuštěním vlastního operačního systému, a pak je využívá po celou dobu své činnosti. Není tedy třeba využívat uvolňování paměti ani opětovnou alokaci. V [77] je dále uvedeno, že tato implementace je vždy deterministická.

Implementace alokuje pole o konfigurovatelné velikosti, které představuje haldu (*heap*) a při potřebě alokovat paměť toto pole dále dělí. Podle [60] je toto řešení velmi paměťově náročné.

Tuto variantu správy paměti lze nalézt v souboru **heap_1.c** v adresáři **MemMang**.

Varianta 2

Na rozdíl od předchozí varianty, tato umožňuje také uvolňování paměti. Uvolněná paměť však zůstává v blocích, není možné spojit přilehlé volné bloky do jednoho. V [77] je uvedeno, že tato varianta již není deterministická, avšak nabízí lepší efektivitu než standardní funkce. Implementace používá k alokaci algoritmus *best fit*, který vyhledává nejmenší volný blok, vyhovující svou velikostí požadavku alokace [78]. Příklad takové alokace je uveden v [60].

Vzhledem k uvedeným vlastnostem je tato implementace vhodná pro aplikace, které dynamicky alokují paměť, avšak velikost alokované paměti je vždy stejná. Jinak může docházet k problémům s fragmentací paměti. Pokud je např. dynamicky alokována fronta, která má při každé alokaci jinou délku, vzniká ve volné paměti množství malých samostatných bloků, které mohou znemožnit další alokaci [77].

Tuto variantu lze nalézt v souboru **heap_2.c** v adresáři **MemMang**.

Varianta 3

Tato varianta se od předchozích výrazně liší. V podstatě se nejedná o vlastní implementaci správy paměti, ale pouze o speciální využití standardních funkcí `malloc()` a `free()`. Implementace definuje jednoduchou funkci (tzv. *wrapper*), která pouze vhodným způsobem volá standardní funkce. Tato funkce zajistí bezpečné provedení (tzv. *thread safety*) alokace paměti. Toho je dosaženo pozastavením plánovače v době alokace, popř. uvolňování paměti [60]. Provedení této funkce lze nalézt ve zdrojovém souboru **heap_3.c** v adresáři **MemMang**.

Použití této implementace vyžaduje nastavení haldy pomocí linkeru (nastavení v konfiguračním souboru systému FreeRTOS nemá v tomto případě význam). Samozřejmostí je nutnost podpory použitých standardních knihovních funkcí. V [77] i [60] je dále uvedeno, že tato varianta je také nedeterministická a přináší s sebou podstatné zvětšení velikosti jádra systému.

Varianta 4

Poslední implementace používá alokační algoritmus *first fit*. Tento algoritmus použije k alokaci první volný blok paměti, který svou velikostí splňuje požadavek [78]. Na rozdíl od druhé varianty (**heap_2.c**) zde dochází ke spojování přilehlých volných bloků paměti. Díky tomu je oproti druhé variantě mnohem méně pravděpodobné, že paměť bude fragmentována i při náhodné alokaci [77]. Tato implementace je rovněž nedeterministická.

V [77] je dále uvedeno, že je tato varianta zvláště vhodná pro aplikace, které potřebují přímo volat alokační funkce (namísto nepřímého volání s použitím API funkcí).

Tato implementace byla přidána do systému FreeRTOS verze 7.2.0 v roce 2012 [58] a není tedy uvedena v [60] (rok vytvoření souboru 2009). Lze ji nalézt ve zdrojovém souboru **heap_4.c** v adresáři **MemMang**.

5.7.1 Podpora MPU

Systém FreeRTOS nabízí podporu jednotky pro správu a ochranu paměti (MPU) pro mikrokontroléry s jádrem ARM Cortex-M3. Jednotku MPU lze použít k ochraně jádra systému a k ochraně dat. Také je možné ji použít k ochraně periferií a k detekci přetečení zásobníků úloh [79]. Podpora jednotky MPU pro mikrokontroléry s jádrem ARM Cortex-M4/M4F není prozatím oficiálně oznámena.

5.8 Koprogramy

Kromě klasických úloh (*task*) podporuje systém FreeRTOS také koprogramy (*co-routines*), které jsou volitelnou součástí systému. Lze je použít společně s úlohami, nebo i samostatně namísto úloh. Koprogramy jsou určeny pro použití na malých mikrokontrolérech, které disponují velmi malým množstvím paměti RAM [80]. Koprogramy jsou principiálně obdobné jako úlohy, avšak liší se v několika bodech.

Všechny koprogramy v aplikaci využívají společný zásobník. To přináší výrazné snížení spotřeby paměti RAM oproti použití úloh. Při blokování koprogramu však hrozí ztráta dat, proto je třeba proměnné, které mají být zachovány, deklarovat jako statické.

Koprogramy jsou spouštěny v kooperativním módu a lze jim nastavit prioritu. Tato priorita je však respektována pouze v rámci koprogramů. To znamená, že pokud jsou spouštěny spolu s úlohami, pak mají vždy vyšší prioritu úlohy. Více informací o koprogramech je možné nalézt v [81].

6 OPERAČNÍ SYSTÉM CHIBIOS/RT

ChibiOS/RT je otevřený a volně šiřitelný operační systém reálného času, navržený pro *embedded* aplikace, které požadují výpočetní efektivitu a nízkou paměťovou náročnost.

Mezi hlavní vlastnosti tohoto systému patří kompaktnost, dobrá přenositelnost a vysoká rychlost. Podle [24] dosahuje systém ChibiOS/RT nejlepší doby přepnutí kontextu ve své třídě díky optimalizaci architektury. V [82] je uvedeno, že s mikrokontrolérem z rodiny STM32F4, taktovaném na 168 MHz, a při použití překladače GCC ve verzi 4.6.2 je možné dosáhnout doby přepnutí kontextu 0,4 μs. Současně je uvedena velikost přeloženého jádra systému 6172 B.

Operační systém ChibiOS/RT oficiálně podporuje několik procesorových architektur. Jejich počet je výrazně menší než u dříve popsaného systému FreeRTOS, podle [83] je však hlavním cílem projektu poskytnout dobrou podporu pro několik populárních architektur namísto jakékoliv podpory pro co nejvíce architektur. Výsledkem toho je jednak podpora různých vývojových desek (včetně STM32F4-Discovery) a inicializačních procedur, a zejména možnost využít kompletní vrstvy abstrakce hardwaru (HAL), která nabízí ovladače pro velké množství zařízení, např. sériové rozhraní, A/D převodníky, sběrnice CAN I^2C a SPI, UART, USB atd. Systém lze díky tomu použít jako kompletní operační systém, bez nutnosti jakékoliv komunikace se samotnou hardwarovou částí. Samozřejmostí je možnost využití pouze samotného jádra.

Plánovač jádra (*scheduler*) je preemptivní, mezi synchronizační a komunikační prostředky patří semafory, mutexy, podmíněné proměnné, synchronní a asynchronní zprávy, příznaky událostí a fronty. Systém nabízí také softwarové časovače.

V [84] jsou dále uvedeny některé vlastnosti systému ChibiOS/RT. Jednou z těchto vlastností je, že jádro systému využívá pouze statickou alokaci paměti, tj. alokace probíhá při překladu. Je možné využít i dynamickou alokaci, ta je však pouze jako volitelná vrstva nad staticky alokovaným jádrem.

Další vlastností je, že jádro nevyužívá žádné tabulky a pole s pevnou velikostí. Díky tomu nedojde k přetečení.

Systém nabízí jednoduché aplikační rozhraní, API funkce mají pouze jeden účel a neobsahují testování parametrů. Tím nedochází ke zpomalování provádění těchto funkcí. Testování parametrů však může být aktivováno při ladění. Podle [84] jsou API funkce vytvořeny tak, aby při předání správných parametrů nikdy nedošlo k selhání. Výjimkou jsou volitelné funkce dynamické alokace, které mohou nahlásit nedostatek paměti.

Lze využít také podpory externích knihoven pro práci se souborovým systémem a síťovou komunikací. Tyto knihovny jsou součástí distribuce systému. Dále je možné využít grafickou knihovnu μGFX (www.ugfx.org), která byla původně určena pouze pro tento systém.

Hlavním vývojářem operačního systému ChibiOS/RT je Giovanni Di Sirio, který je také vedoucím projektu. Samotný systém ChibiOS/RT je vyvíjen od roku 2007, avšak je založen na systému, který vznikl kolem roku 1992 [85], [86].

Systém ChibiOS/RT je ve své stabilní verzi šířen pod licencí GNU GPL3 s výjimkou. Tato výjimka, podobně jako u systému FreeRTOS, povoluje za určitých podmínek použití v aplikacích, které obsahují také uzavřený kód. Dále je možné získat i několik komerčních licencí. Více informací lze nalézt v [87].

6.1 Podporované architektury a nástroje

Jak bylo výše uvedeno, operační systém ChibiOS/RT podporuje poměrně malé množství procesorových architektur a jeho vývoj je směřován především k dobré podpoře těchto vybraných architektur. Seznam oficiálně podporovaných rodin mikrokontrolérů a překladačů je uveden v Tab. 6-1, převzaté z [83]. Z této tabulky je patrné, že rozsah podporovaných mikrokontrolérů je výrazně menší než u dříve popsaného systému FreeRTOS.

V současné době je podporováno celkem osm různých architektur. V tabulce lze nalézt také rodinu STM32F4 výrobce ST Microelectronics, založenou na jádru ARM Cortex-M4. Pro tuto rodinu podporovány překladače GCC, IAR a RVCT. Součástí distribuce je také demonstrační aplikace pro mikrokontrolér STM32F407 a vývojovou desku STM32F4 Discovery).

Tab. 6-1: Architektury podporované operačním systémem ChibiOS/RT [83]

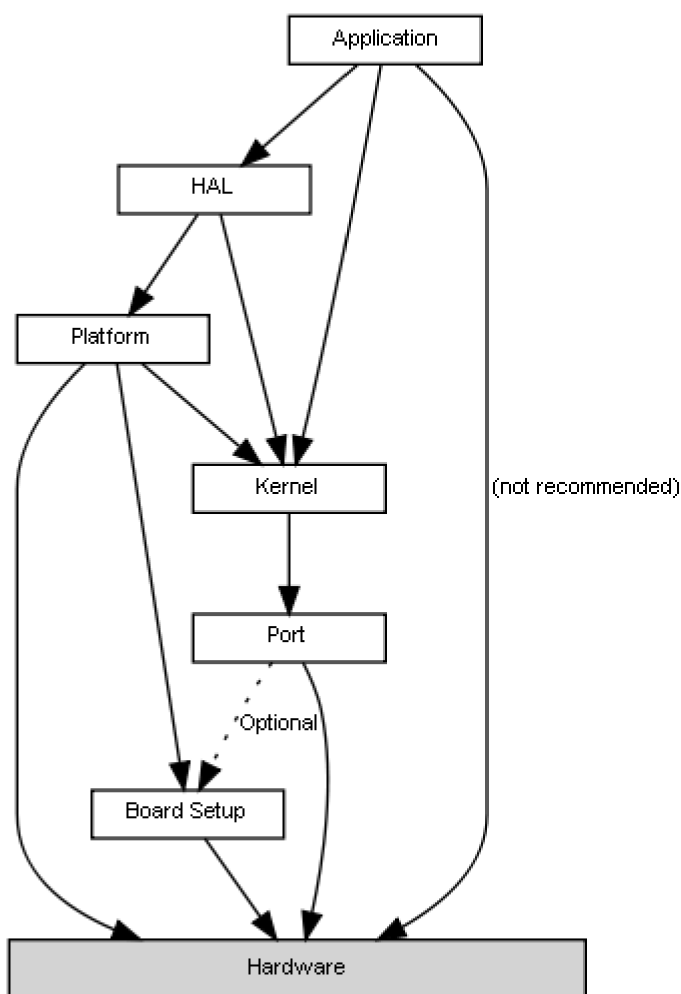
architektura	překladač	podporované rodiny mikrokontrolérů
ARM Cortex-M0	GCC	LPC11, LPC11U, STM32F0
ARM Cortex-M0	RVCT	LPC11, LPC11U, STM32F0
ARM Cortex-M3	GCC	LPC13, STM32F1, STM32F2, STM32L1
ARM Cortex-M3	IAR	LPC13, STM32F1, STM32F2, STM32L1
ARM Cortex-M3	RVCT	LPC13, STM32F1, STM32F2, STM32L1
ARM Cortex-M4	GCC	STM32F3, STM32F4
ARM Cortex-M4	IAR	STM32F3, STM32F4
ARM Cortex-M4	RVCT	STM32F3, STM32F4
ARM7	GCC	AT91SAM7, LPC214
MegaAVR	GCC	ATmega128, AT90CAN128, ATmega328p, ATmega1280
MSP430	GCC	MSP430F1611
Power Architecture e200z	GCC/HighTec	SPC56
STM8	Cosmic	STM8L, STM8S
STM8	Raisonance	STM8L, STM8S

V [88] lze nalézt dvě přehledné tabulky, které ukazují podporované subsystémy jádra pro jednotlivé architektury a dále podporu subsystémů vrstvy HAL pro různé rodiny mikrokontrolérů. Také je zde uveden simulátor pro architekturu x86.

6.2 Architektura systému

Operační systém ChibiOS/RT je velmi modulární a je rozdělen do několika nezávislých součástí. Tyto součásti jsou dále děleny na subsystémy. Hlavní části systému jsou popsány dále. Mimo tyto části jsou také nabízeny některé další funkce [89]. Na Obr. 6-1 jsou znázorněny vztahy mezi jednotlivými součástmi systému.

Obr. 6-1: Vztahy jednotlivých součástí operačního systému ChibiOS/RT [89]

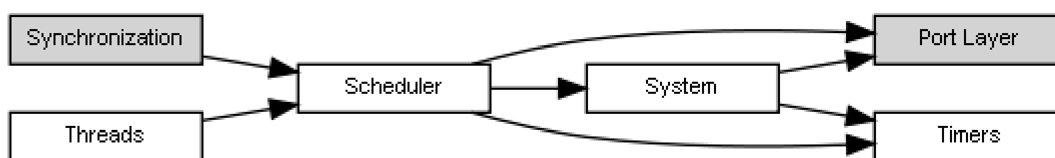


Jádro systému (*Kernel*)

Základní část systému, nezávislá na použité platformě. Spolu s vrstvou *port layer* tvoří samostatně funkční celek, schopný funkce nezávisle na ostatních částech systému.

Samotné jádro je taktéž modulární a je rozděleno na několik subsystémů. Většina z těchto subsystémů je volitelná a jejich použití není nutné. Vazby mezi subsystémy jádra jsou znázorněny na Obr. 6-2. Základními a povinnými subsystémy jsou inicializace a nízkoúrovňová správa systému (*System*), plánovač (*Scheduler*), správa úloh (*Threads*) a časování (*Timers*). Jejich popis lze nalézt v [90]. Mimo tyto povinné subsystémy je zde uvedena také skupina subsystémů pro synchronizaci (synchronizační mechanismy jsou popsány podrobněji v části 6.6) a vrstva *port layer* (viz dále). Součástí jádra systému je pak také správa paměti (část 6.7) a dále vstupně/výstupní mechanismy (více také v [91]) a prostředky pro ladění [92], [93].

Obr. 6-2: Vztahy mezi subsystémy jádra systému ChibiOS/RT [89]



Vrstva *port layer*

Tato vrstva patří k hlavním součástem systému. Je závislá na dané architektuře, resp. kombinaci architektury a použitého překladače. Port vrstva definuje inicializační procedury systému, abstrakci vektorů přerušení, zámky systému, přepínání kontextu. Díky tomu se jedná o velmi kritickou část operačního systému, jejíž implementace může mít velký vliv na celkový výkon systému.

Platformní vrstva (*Platform layer*)

Platformní vrstva obsahuje implementaci nízkoúrovňových ovladačů a obvykle odpovídá jednotlivým podporovaným rodinám mikrokontrolérů.

Inicializace vývojové desky

Systém také přímo podporuje různé typy vývojových desek pro podporované mikrokontroléry. Tato část je pak využita k inicializaci desky a definici některých vstupně/výstupních vývodů použitého mikrokontroléru, např. připojených LED diod.

Vrstva abstrakce hardwaru (HAL)

Tato vrstva obsahuje ovladače běžných i méně běžných zařízení, většinou komunikačních. Nabízí společné aplikační rozhraní, které je nezávislé na použité platformě a překladači.

Vrstva HAL využívá platformní vrstvu, která, jak bylo uvedeno, poskytuje nízkoúrovňové implementace ovladačů hardwaru. Systém ChibiOS/RT obsahuje dva

typy ovladačů zařízení. Tyto ovladače jsou rozlišeny podle toho, zda komunikují přímo s hardwarem, či nikoliv.

Prvním typem je běžný ovladač zařízení (*Normal Device Driver*). Tento ovladač je rozdělen na část vysokoúrovňovou (*high level driver*, *HLD*), která komunikuje s aplikací, a část nízkoúrovňovou (*low level driver*, *LLD*), která zprostředkovává přístup ke konkrétnímu hardwaru. Vysokoúrovňová část definuje API funkce a části ovladače, které jsou nezávislé na platformě. Nízkoúrovňová část ovladače je pak implementována v platformní vrstvě. Struktura běžného ovladače je na Obr. 6-3. Příkladem běžného ovladače může být ovladač pro komunikační zařízení SPI.

Obr. 6-3: Struktura běžného ovladače zařízení [89]

Druhým typem ovladače je komplexní ovladač zařízení (*Complex Device Driver*). Tento typ ovladače nekomunikuje přímo s hardwarem, ale pouze definuje vysokoúrovňové funkce pro práci se zařízením. Pro přístup k hardwaru pak využívá ostatních ovladačů a to buď běžných nebo komplexních. Jako příklad komplexního ovladače lze uvést ovladač komunikace pro paměťové karty MMC/SD, který využívá běžný ovladač rozhraní SPI [89]. Více informací o vrstvě HAL a jednotlivých poskytovaných ovladačích zařízení lze nalézt v [94].

6.3 Struktura zdrojových souborů

Distribuce operačního systému ChibiOS/RT obsahuje zdrojové soubory jádra systému, částí specifických pro danou architekturu, vrstvy HAL a dalších volitelných součástí. Další součástí distribuce je dokumentace, množství demonstračních a testovacích aplikací a také externí knihovny pro síťové rozhraní a souborový systém.

Zdrojové soubory operačního systému ChibiOS/RT je možné získat několika způsoby [95]. Nejjednodušším způsobem je stažení zip archivu. Stabilní verze systému vždy obsahuje sudé číslo v označení verze, např. 2.6.3, která je aktuální v době psaní této práce.

Ve staženém adresáři **ChibiOS_2.6.3** lze nalézt textové soubory s popisem distribuce a historií a také licenční podmínky. Dále obsahuje několik podadresářů. Jsou to podadresáře **boards**, kde jsou umístěny konfigurační a zdrojové soubory pro různé vývojové desky, **demos**, obsahující demonstrační aplikace, **docs**, obsahující soubory dokumentace, **ext**, kde lze nalézt externí knihovny, **os**, který obsahuje zdrojové soubory jádra a dalších součástí systému, **test**, kde se nachází testovací aplikace pro systém a **testhal**, obsahující aplikace pro testování vrstvy HAL.

Následuje popis struktury adresáře **os**, který je nejdůležitější. Tento adresář obsahuje čtyři podadresáře. V podadresáři **kernel** jsou umístěny zdrojové a hlavičkové soubory jádra systému. Podadresář **ports** obsahuje zdrojové soubory pro jednotlivé podporované architektury. Podobně jako u systému FreeRTOS je nejprve provedeno rozdělení do podadresářů podle překladače a následně podle architektury. V podadresáři **hal** jsou umístěny zdrojové soubory a dokumentace vrstvy HAL. Poslední podadresář **various** obsahuje některé další funkce a knihovny, které nejsou přímo součástí systému [96].

Konfigurace systému je provedena pomocí konfiguračních hlavičkových souborů. Ty jsou rozděleny podle částí systému. Základním konfiguračním souborem, který je vyžadován jádrem systému, je **chconf.h**, který lze nalézt v adresáři **ChibiOS_2.6.3/os/kernel/templates** a také v adresáři každé demonstrační aplikace. Zde lze konfigurovat základní parametry systému jako je frekvence systémových hodin, velikost přidělené paměti RAM, optimalizace rychlosti. Dále povolování všech volitelných částí systému, např. jednotlivých synchronizačních mechanismů a také nastavení ladicích prostředků [97].

Pro využití vrstvy HAL je dále třeba použít další konfigurační soubor **halconf.h**, který je umístěn v adresáři **ChibiOS_2.6.3/os/hal/templates**. Obsahuje vysokoúrovňová nastavení jednotlivých ovladačů zařízení včetně možnosti volby, které ovladače mají být použity. Nízkoúrovňová nastavení, závislá na použité platformě se nachází v konfiguračním souboru **mcuconf.h**. Tento soubor lze nalézt v adresáři demonstračních aplikací, vždy pro konkrétní typ mikrokontroléru [98].

6.4 Správa úloh

V dokumentaci k operačnímu systému ChibiOS/RT je místo výrazu úloha (*task*) používán výraz vlákno (*thread*). Vzhledem k tomu, že v tomto případě jde v podstatě pouze o věc používání názvosloví a nikoli rozdílu ve funkci, lze oba tyto názvy považovat za ekvivalentní.

Operační systém ChibiOS/RT nabízí dva typy úloh. Jsou to úlohy statické a dynamické. Výchozím typem jsou statické úlohy, které vyžadují staticky alokovanou oblast paměti. Paměť pro dynamické úlohy může být alokována dvěma způsoby a to buď v haldě (*heap*), nebo po blocích o pevné velikosti (*memory pool*). Mechanismy správy paměti jsou stručně popsány v části 6.7. Více informací o možnostech vytváření úloh lze nalézt v [99] a [100].

Každá úloha v systému ChibiOS/RT je představována strukturou *Thread Structure*, která obsahuje informace potřebné ke správě úloh. V Tab. 6-2 je uvedena její podoba. Tato struktura je definována v hlavičkovém souboru **chthreads.h**, který lze nalézt v adresáři **ChibiOS_2.6.3/os/kernel/include**. Data uvedená v Tab. 6-2 byla převzata

přímo z tohoto souboru. Podobně jako v systému FreeRTOS je většina těchto položek definována podmíněně v závislosti na konfiguraci systému.

Tab. 6-2: Struktura reprezentující úlohu v systému ChibiOS/RT

název proměnné	význam
p_next	ukazatel další úlohy v seznamu/frontě
p_prev	ukazatel předcházející úlohy v seznamu/frontě
p_prio	priorita úlohy
p_ctx	struktura představující kontext
p_newer	ukazatel novější úlohy v registru
p_older	ukazatel starší úlohy v registru
p_name	název úlohy
p_stklimit	ukazatel konce zásobníku
p_state	aktuální stav úlohy
p_flags	různé příznaky úlohy
p_refs	odkazy na danou úlohu
p_preempt	čas, který zbývá úloze do preempce (Round-Robin)
p_time	čas spotřebovaný úlohou
rdymsg	kód předaný při přechodu do stavu Připravená [4]
exitcode	ukončovací kód [5]
wtobjp	ukazatel na synchronizační objekt, na který úloha čeká
ewmask	maska povolených událostí
p_waiting	seznam čekajících úloh
p_msgqueue	fronta pro zprávy
p_msg	zpráva
p_epending	maska čekajících událostí
p_mtxlist	seznam mutexů, držených danou úlohou
p_realprio	vlastní priorita úlohy, nezděděná
p_mpool	memory pool, kde se nachází pracovní oblast úlohy
THREAD_EXT_FIELDS	další položky [6]

Popsaná struktura představuje obdobu TCB, popsaného v kapitole 5 pro systém FreeRTOS. Každé úloze náleží mimo tuto strukturu také vlastní zásobník a několik dalších oblastí, určených k přepínání kontextu a obsluze přerušení. Vše je alokováno do

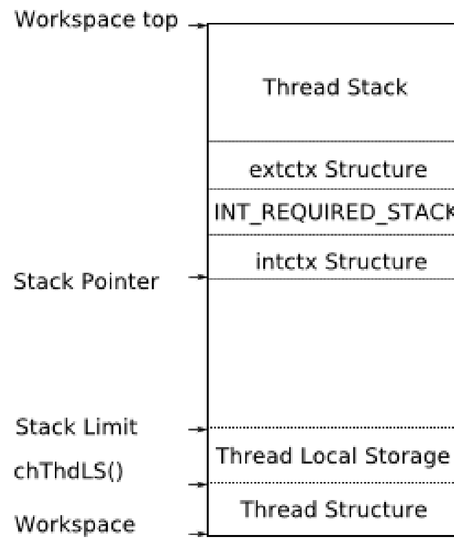
[4] Zpráva, která je poslána úloze při jejím zavolání z jiné úlohy nebo obslužné rutiny přerušení.

[5] Kód, který je proveden při ukončení/pozastavení úlohy jinou úlohou.

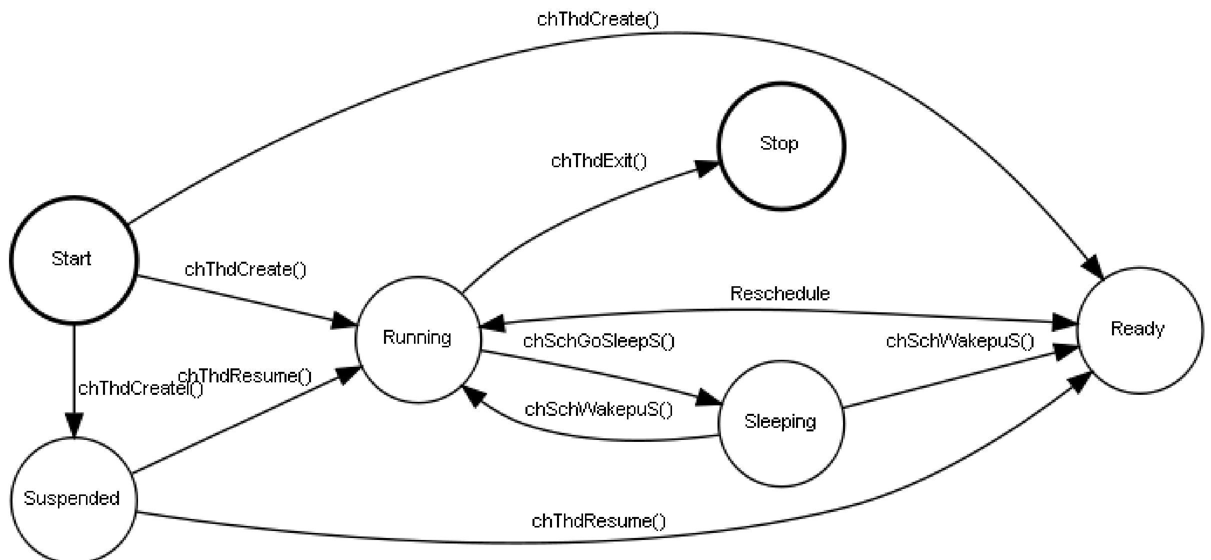
[6] Tyto položky nejsou obvykle přítomné, lze je libovolně definovat uživatelsky v konfiguračním souboru **chconf.h** (viz část 6.3).

oblasti paměti nazvané *Thread Working Area* nebo *Workspace*, což lze přeložit jako pracovní oblast úlohy/vlákna. Pracovní oblast je soukromá pro každou úlohu a daná úloha obvykle nevyužívá paměť mimo tuto oblast. Výjimkou je přístup ke sdíleným datům [101]. Na Obr. 6-4 je znázorněna pracovní oblast úlohy.

Obr. 6-4: Pracovní oblast úlohy [101]



Obr. 6-5: Přechodový diagram stavů úloh v systému ChibiOS/RT [101]



V [101] je uveden přechodový diagram stavů úlohy. Tento diagram je na Obr. 6-5. Z diagramu lze vyčíst, že úloha může nabývat pěti stavů. Těmito stavy jsou

Běžící (*Running*)
Připravená (*Ready*)
Čekající/Spící (*Sleeping*)
Pozastavená (*Suspended*)
Zastavená (*Stop*)

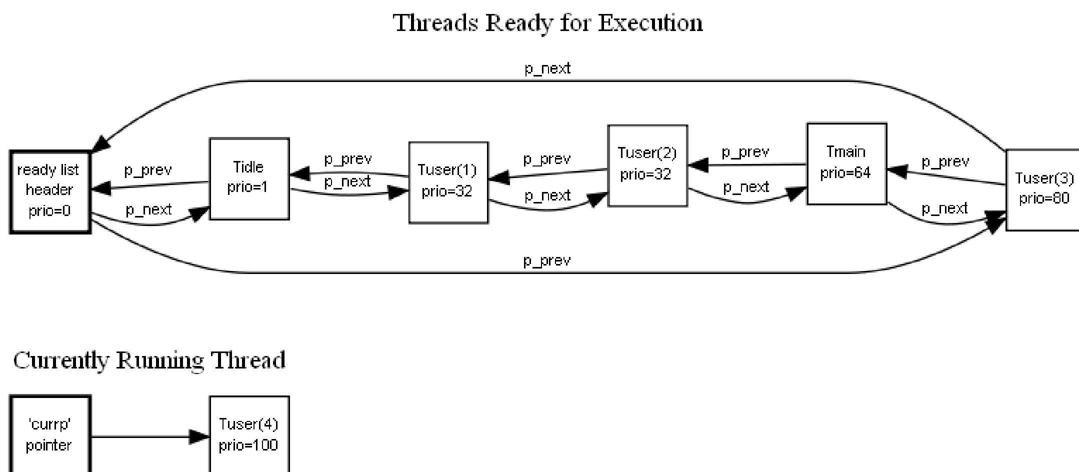
V [90] lze nalézt podstatně větší množství stavů, které představují čekání na různé synchronizační mechanismy, např. semafory, fronty a události.

6.5 Plánování a priority

Plánovač operačního systému ChibiOS/RT je preemptivní. Strategie plánování je založena na prioritách, tj. vždy je spuštěna úloha, která má nejvyšší prioritu a současně se nachází ve stavu Připravená. Pro tento účel systém definuje seznam připravených úloh jako obousměrný seznam, který obsahuje úlohy řazené podle priority [101].

Na Obr. 6-6 je znázorněno propojení mezi jednotlivými položkami tohoto seznamu. Každá položka obsahuje ukazatel na další a také na předcházející položku s tím, že poslední položka seznamu (úloha s nejvyšší prioritou, popř. jedna z těchto úloh) ukazuje zpět na začátek seznamu. Úlohy, které jsou do tohoto seznamu přidávány, jsou vždy zařazeny až za úlohy s vyšší nebo stejnou prioritou. Jak je na Obr. 6-6 dále znázorněno, aktuálně běžící úloha se v seznamu připravených úloh nenachází, k odkazování na tuto úlohu slouží systémový ukazatel s názvem `currp`.

Obr. 6-6: Seznam připravených úloh [101]



Po inicializaci systému jsou automaticky vytvořeny dvě úlohy. První je prázdná úloha (*Idle thread*). Tato úloha má nejnižší prioritu a běží pouze, pokud není prováděna žádná jiná úloha. Druhou vytvořenou úlohou je hlavní úloha (*Main thread*). Tato úloha provádí kód funkce **main()** a obvykle vytváří ostatní úlohy [99].

Systém ChibiOS/RT dovoluje použití 128 úrovní priority s tím, že stejné priority může nabývat více úloh [101]. Pokud má více úloh stejnou prioritu, je pro jejich spouštění použit plánovací algoritmus Round-Robin, podobně jako u systému FreeRTOS. Tento algoritmus byl již popsán v části 5.4. Na rozdíl od systému FreeRTOS, kde je časové kvantum pevně dáno, lze v systému ChibiOS/RT tuto dobu nastavit libovolně v násobcích periody systémových hodin. Pokud je časové kvantum nastaveno jako nulové, dojde k zákazu časového přepínání a úlohy jsou spouštěny v kooperativním režimu. To však platí pouze pro úlohy se stejnou prioritou a úlohy s vyšší prioritou jsou i dále spouštěny preemptivně [97].

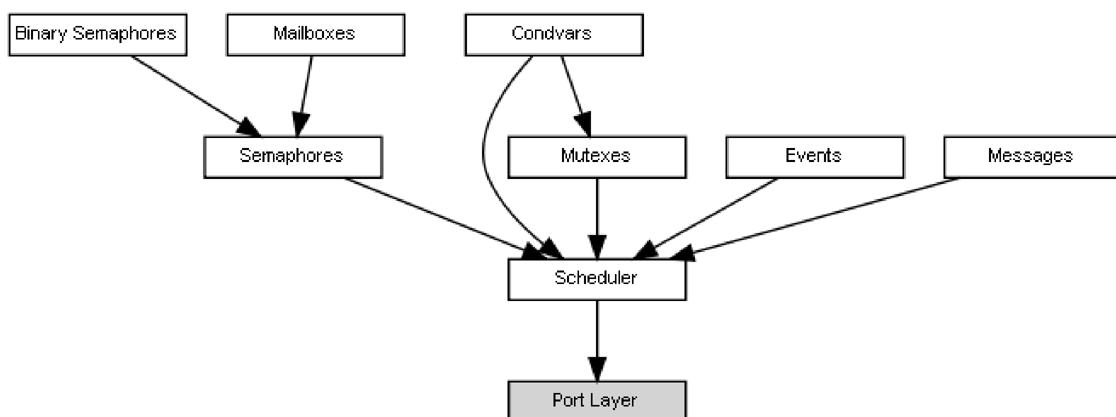
V [102] je uvedeno, že obslužné rutiny přerušení je vhodné považovat za speciální úlohy, jejichž priorita bude nastavena nad oblastí priorit, určených pro běžné úlohy. Toto je obzvláště vhodné při použití v architekturách, které umožňují preempci obslužných rutin přerušení (např. architektura ARM Cortex-M).

6.6 Synchronizace a komunikace

Jak bylo uvedeno, systém ChibiOS/RT pro synchronizaci úloh a komunikaci nabízí poměrně velké množství prostředků. Kromě základních prostředků jako jsou semafory nebo fronty lze využít také synchronních a asynchronních zpráv a podmíněných proměnných.

Synchronizační mechanismy s výjimkou binárních semaforů, asynchronních zpráv a podmíněných proměnných využívají pouze API funkcí plánovače. Schematické znázornění vazeb mezi synchronizačními prostředky a plánovačem jádra je na Obr. 6-7.

Obr. 6-7: Vztahy mezi synchronizačními prostředky [89]



6.6.1 Čítací semafory

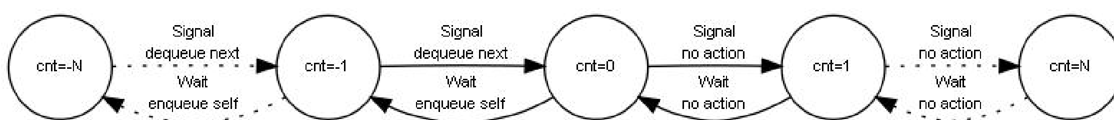
Tento synchronizační prostředek může nabývat libovolného množství stavů. Základem čítacího semaforu je čítač, tj. proměnná, která v podstatě určuje stav semaforu. Tato proměnná může nabývat také záporných hodnot. V případě, že je hodnota záporná, udává stav čítače počet úloh, čekajících na semafor.

Pro čítací semafory jsou definovány dvě operace. První operací je čekání (*wait*). Prakticky jde obdobu operace *take*, která byla popsána v části 5.5.2. Při provedení operace čekání je dekrementována hodnota čítače. Pokud je po dekrementaci hodnota záporná, je obsluhovaná úloha zařazena do čekací fronty. Druhou operací je signál (*signal*). Opět se jedná o obdobu operace *give* u semaforů v systému FreeRTOS. Operace signál inkrementuje čítač semaforu. Pokud je výsledkem nezáporná hodnota, je volající úloha odstraněna z čekací fronty. Na Obr. 6-8 je přechodový diagram stavů čítacího semaforu. Uvedené operace mohou být prováděny libovolnou úlohou, semafory nejsou na rozdíl od mutexů vlastněny žádnou úlohou.

Jsou definována i makra pro rychlou změnu nebo zjištění stavu semaforu [103].

Čítací semafor lze vytvořit s libovolnou nezápornou počáteční hodnotou. Při operacích se semafory lze využít časové limity. Nastavením časového limitu je možné zajistit určitou dobu čekání úlohy na semafor. Pokud není během této doby semafor k dispozici, dochází k odstranění úlohy z čekací fronty a je obnoveno její vykonávání. Také je možné semafor resetovat, čímž dojde k vyprázdnění fronty čekajících úloh. Další funkcí je předávání zpráv při provádění operace čekání. Tato zpráva nese informaci, zda operace proběhla normálním způsobem nebo během čekání nastala nějaká událost, např. vypršel nastavený časový limit [104].

Obr. 6-8: Přechodový diagram stavů čítacího semaforu [104]



V [104] je dále uveden příklad použití čítacího semaforu při hlídání zdrojů. Také je zde uvedena poznámka, že pomocí mechanismu čítacích semaforů jsou implementovány vstupně/výstupní fronty.

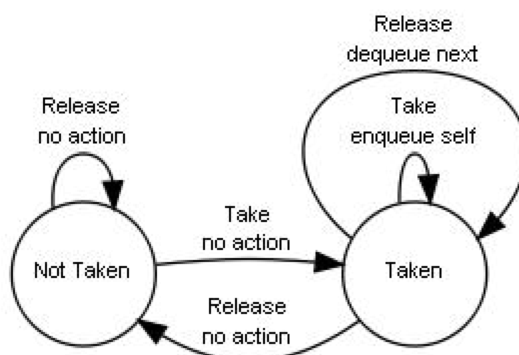
6.6.2 Binární semafory

Binární semafory využívají funkcionalitu čítacích semaforů a jsou implementovány v podobě maker, která využívají funkce definované pro čítací semafory. Obsah čítače binárního semaforu nemůže překročit hodnotu 1. Binární semafor má tedy pouze dva stavy, které určují, zda je tento semafor k dispozici, či nikoliv. Hodnota čítače může být záporná a v tomto případě opět udává počet úloh čekajících na semafor. Pokud je

hodnota čítače záporná nebo nulová, semafor není k dispozici (stav *taken*), pokud je rovna jedné, je semafor k dispozici (stav *not taken*) [105].

Podobně jako pro čítací semafory, jsou i pro binární semafory definovány dvě operace. První operací je přečtení semaforu (*take*). Tato operace převádí semafor do stavu *taken*. Pokud je již v tomto stavu, je volající úloha zařazena do seznamu čekajících. Druhou operací je uvolnění (*release*). Tato operace má za následek přechod semaforu do stavu, kdy je k dispozici (*not taken*), popř. odstranění volající úlohy z čekací fronty. Pokud je operace uvolnění provedena na semaforu, který již byl uvolněn (je k dispozici, tj. ve stavu *not taken*), nemá tato operace žádný efekt. Stejně jako u čítacích semaforů může s binárními semafory pracovat kterákoli úloha nebo obslužná rutina přerušení. Binární semafory mohou být vytvořeny s libovolným počátečním stavem a lze využít časové limity a reset stejně jako u čítacích semaforů. Taktéž jsou k dispozici zprávy o stavu operace čekání [104]. Na Obr. 6-9 je přechodový diagram stavů binárního semaforu.

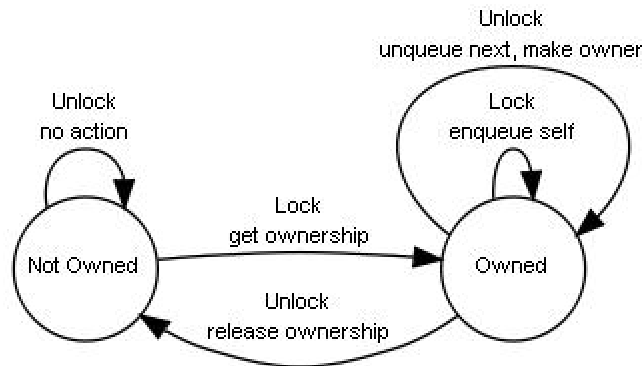
Obr. 6-9: Přechodový diagram stavů binárního semaforu [104]



6.6.3 Mutexy

Mezi synchronizační prostředky patří také mutexy, které již byly popsány v části 5.5.4. Na rozdíl od semaforů mají mutexy vlastníka. Tak jako binární semafory, mohou mutexy nabývat dvou stavů. Tyto stavy určují, zda je mutex vlastněný (*owned*), či volný (*not owned*). Úlohy mohou s mutexy provádět dva druhy operací. První operací je uzamknutí (*lock*). Úloha, která tuto operaci provede, získává vlastnictví, popř., pokud je mutex již vlastněn jinou úlohou, je volající úloha přesunuta do fronty čekajících úloh. Druhou operací je odemknutí mutexu (*unlock*). Tuto operaci může provést pouze úloha, která vlastní mutex. Pokud je v čekací frontě nejaká úloha, je tato z fronty odstraněna a je jí předáno vlastnictví mutexu [104]. Na Obr. 6-10 je přechodový diagram stavů mutexu.

Obr. 6-10: Přechodový diagram stavů mutexu [104]



Podle [106] má funkcionalita mutexů v systému ChibiOS/RT implementovanou plnou podporu dědění priorit k minimalizaci problému inverze priorit. Při zařazení úlohy do fronty úloh, čekajících na mutex, je úloze, která mutex vlastní zvýšena priorita na hodnotu, kterou má úloha čekající. Samozřejmě to platí pouze v případě, že čekající úloha má vyšší prioritu než úloha, která vlastní mutex. Pokud má prioritu nižší nebo stejnou, priorita úlohy, vlastnící mutex, se nemění. Tento mechanismus pracuje s libovolným množstvím mutexů a úloh.

V [104] je dále uvedeno několik poznámek k používání mutexů v systému ChibiOS/RT. Je zde uvedeno, že operace odemknutí (*unlock*) nepřijímá žádné parametry a při volání této operace systém vždy odemkne poslední mutex, který daná úloha vlastní. Pro tyto účely je pro každou úlohu spravován seznam vlastněných mutexů (viz Tab. 6-2). Navíc je definována operace pro odemknutí všech mutexů, vlastněných úlohou (*unlock all*). Další přídavnou operací je pokusné uzamknutí mutexu (*try lock*), kdy při neúspěšném pokusu o uzamknutí (získání vlastnictví) mutexu nedochází k čekání.

6.6.4 Podmíněné proměnné

Podmíněné proměnné v systému ChibiOS/RT jsou implementovány jako rozšíření funkcionality mutexů a nemohou být použity samostatně. Použitím mutexů a podmíněných proměnných lze implementovat synchronizační prostředek, zvaný monitor [107].

Podmíněná proměnná slouží k signalizaci splnění nějaké podmínky a neobsahuje tedy žádnou hodnotu. Podmíněná proměnná je představována frontou úloh, čekajících na tuto proměnnou, tj. čekajících na splnění určité podmínky [108]. Úloha, čekající na splnění podmínky je zablokována, přidána do čekací fronty podmíněné proměnné a mutex, aktuálně vlastněný touto úlohou, je odemknut. Po splnění podmínky je mutex opět uzamknut a čekající úloha je odblokována.

6.6.5 Příznaky událostí

Správa událostí je důležitou součástí operačního systému ChibiOS/RT. Většina nízkoúrovňových ovladačů generuje události pro signalizaci stavu aplikaci. Události jsou generovány asynchronně.

Systém ChibiOS/RT implementuje správu událostí specifickým způsobem a definuje dva objekty pro práci s událostmi. Prvním objektem je zdroj událostí (*Event Source*). Jedná se o systémový objekt, který slouží k upozornění na událost. Pokud událost nastane, může úloha nebo obslužná rutina přerušení vyslat (*broadcast*) pomocí zdroje události informaci o příchodu této události všem registrovaným úlohám. V systému může existovat neomezené množství zdrojů událostí [109].

Dalším objektem je sledovač událostí (*Event Listener*). Tento objekt slouží k propojení úloh se zdroji událostí. Každá úloha může sledovat neomezené množství těchto zdrojů. Proces přiřazení úlohy ke zdroji událostí pomocí sledovače událostí se nazývá registrace (*registration*) [110]. Každý sledovač událostí má vlastní masku, která je logicky přičtena k masce příslušné úlohy. To znamená, že každé úloze lze nezávisle definovat identifikátory událostí. Je možné, aby úloha čekala na jednu nebo více událostí a dále je možné nastavit logické podmínky pro toto čekání.

6.6.6 Zprávy

Operační systém ChibiOS/RT nabízí dva druhy přenosu zpráv mezi úlohami. Prvním je synchronní přenos, kdy daná úloha vysílá zprávu určenou pro jinou konkrétní úlohu, druhým je asynchronní přenos, pomocí kterého může mezi sebou komunikovat více úloh přes frontu zpráv.

Synchronní zprávy (*synchronous messages*) poskytují jednoduchý a rychlý mechanismus pro komunikaci mezi úlohami. Tento mechanismus dovoluje přenos zpráv dvěma směry. Velké rychlosti je dosaženo tím, že mezi úlohami nejsou kopírována samotná data, ale pouze dojde k přenosu ukazatele na tato data. Zprávy jsou obvykle zpracovávány v pořadí podle příchodu (FIFO), je však možné je zpracovávat také podle priorit. Vykonávání úlohy, čekající na zprávu je pozastaveno až do příchodu této zprávy. Úloha, která zprávu vyšle, je taktéž zastavena, dokud nedostane od přijímající úlohy oznámení o přijetí. Více informací o synchronních zprávách lze nalézt v [111].

Mechanismus asynchronních zpráv využívá jednosměrnou frontu zpráv a objekt *mailbox*. Tento objekt obsahuje odkaz na frontu a také semafory, jejichž funkcionalita je využita k signalizaci míry zaplnění fronty. Pokud je fronta plná, úloha, která vysílá zprávu, čeká, dokud není místo uvolněno, popř. dokud nevyprší nastavený časový limit [112]. Na rozdíl od synchronních zpráv nečeká úloha, odesílající zprávu, na odpověď přijímající úlohy.

6.6.7 Fronty

Fronty jsou v systému ChibiOS/RT využívány většinou v ovladačích zařízení, využívajících sériovou komunikaci. Ovladače jsou obvykle rozděleny na dvě části. Fronty mohou být vstupní (*Input queue*), kdy nízkoúrovňová část ovladače do fronty zapisuje a vysokoúrovňová část čte, nebo výstupní (*Output queue*), kdy do fronty zapisuje vysokoúrovňová část a nízkoúrovňová část z fronty čte. Pokud je zkombinována vstupní a výstupní fronta, vzniká obousměrná fronta (*Full duplex queue*) [113].

6.7 Správa paměti

Jak bylo uvedeno jádro systému ChibiOS/RT využívá statickou alokaci paměti. Díky tomu se lze obejít bez správy paměti, pokud to aplikace dovoluje. Správa paměti je však dostupná v podobě volitelných součástí systému.

Operační systém ChibiOS/RT nabízí tři různé možnosti alokace paměti, které budou popsány dále. Také je možné využít standardních funkcí pro alokaci. V [114] lze nalézt přehlednou srovnávací tabulku pro různé možnosti správy paměti. Na Obr. 6-11 jsou pak znázorněny vazby mezi jednotlivými mechanismy. V obrázku jsou také znázorněny dynamické úlohy (*Dynamic Threads*), které využívají zde popisovaných alokačních prostředků (viz část 6.4). Následuje stručný popis alokačních mechanismů poskytovaných systémem ChibiOS/RT.

Hlavní alokátor paměti (*Core memory manager*, *Core allocator*)

Tato součást pracuje jako hlavní správce paměti. Její funkcionalitu využívají ostatní mechanismy správy paměti. Tím je zajištěna konzistentnost operací nad pamětí a je možné, aby současně pracovalo více alokačních mechanismů.

Jedná se o jednoduchý alokační mechanismus, který umožňuje pouze alokaci bloků paměti bez možnosti jejich uvolnění. Mechanismus je deterministický a bezpečný, lze jej použít také v obslužných rutinách přerušení a v [114] je jeho použití obecně doporučeno, pokud je aplikací vyžadována dynamická alokace bez požadavku na uvolňování paměti. Tento alokátor také poskytuje bloky paměti pro další alokační mechanismy [115].

Alokace v haldě (*Heap allocator*)

Alokační mechanismus pro dynamickou alokaci v haldě včetně možnosti uvolňovat paměť. K alokaci je použit algoritmus *first fit*, který byl stručně popsán v části 6.6.4. Další informace o algoritmu *first fit* lze nalézt v [78]. Mechanismus již není

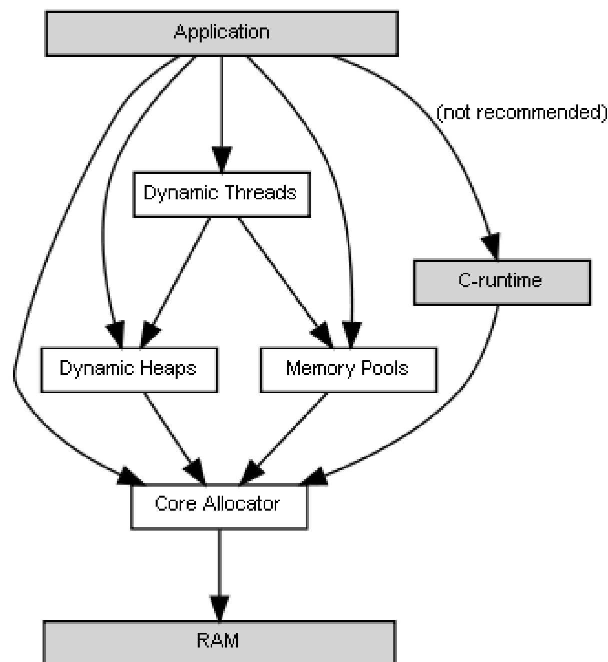
deterministický a přináší nebezpečí fragmentace paměti. Také jej nelze použít v obslužných rutinách přerušení [114].

Funkce definované pro tento mechanismus jsou ekvivalentní k standardním funkcím `malloc()` a `free()`. Hlavním rozdílem oproti standardním funkcím je zajištění bezpečného volání (*thread safety*). Je také možné využít API funkce tohoto alokátoru pouze jako *wrapper* funkce pro standardní alokační funkce [116].

Alokace po blocích (*Memory pools*)

Tento mechanismus nabízí alokaci i uvolňování bloků paměti s pevnou velikostí. Jeho výhodou je především konstantní čas alokace (determinismus) a také absence problémů s fragmentací paměti a vysoká rychlost [117]. Lze jej použít i v obslužných rutinách přerušení a lze také definovat více takových bloků [114].

Obr. 6-11: Správa paměti v systému ChibiOS/RT [89]



7 MOŽNOSTI IMPLEMENTACE PSD REGULÁTORŮ S VYUŽITÍM OS REÁLNÉHO ČASU

Tato kapitola je věnována stručnému obecnému popisu číslicového PSD regulátoru z teoretického i praktického hlediska a možnostem implementace jediného PSD regulátoru a také několika takových regulátorů při použití operačního systému reálného času.

7.1 PSD regulátor

Proporcionálně-sumačně-diferenční regulátor je číslicovou obdobou spojitého PID regulátoru. Čím kratší je vzorkovací perioda, tím více se výstup diskrétního PSD regulátoru blíží výstupu spojitého regulátoru PID. Proto musí být vzorkovací perioda poměrně krátká. Samozřejmostí je splnění vzorkovacího teorému.

Ideální spojitý PID regulátor lze popsat následující rovnicí [119].

$$u(t) = K\left(e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau + T_D\frac{de(t)}{dt}\right) \tag{7-1}$$

kde $u(t)$ je akční zásah (výstup) regulátoru,

K je zesílení regulátoru,

T_I je integrační časová konstanta,

T_D je derivační časová konstanta,

$e(t) = w(t) - y(t)$ je regulační odchylka, tj. rozdíl mezi žádanou hodnotou a výstupem soustavy.

PSD regulátor zpracovává informace v diskrétních časových okamžicích $t = kT$, kde $k = 0,1,2,\ldots$ a T je perioda vzorkování.

Pro diskrétní PSD regulátor je integrační složka nahrazena sumací. Pro náhradu integrace lze použít zpětnou obdélníkovou, dopřednou obdélníkovou nebo lichoběžníkovou náhradu. Nejběžnější náhradou je zpětná obdélníková metoda. Derivační složka je nahrazena zpětnou diferencí [118].

PSD regulátor lze pak popsat následující rovnicí [118].

$$u(kT) = K\left\{e(kT) + \frac{T}{T_I}\sum_{i=1}^{k} e(iT) + \frac{T_D}{T}\left(e(kT) - e[(k-1)T]\right)\right\} \quad (7\text{-}2)$$

Tento tvar se nazývá polohový popř. paralelní [119]. Jeho nevýhodou je podle [118] nutnost uchovávat v paměti všechny hodnoty $e(iT)$, $i = 1,2,\ldots,k$ pro výpočet sumační části.

Zmíněnou nevýhodu odstraňuje přírůstkový tvar regulátoru. Tento tvar je rekurentní a určuje se pouze přírůstek akční veličiny k její předchozí hodnotě. Pro tento přírůstek platí

$$\nabla u(kT) = u(kT) - u[(k-1)T] \quad (7\text{-}3)$$

Dosazením rovnice pro polohový tvar (7-2) do rovnice (7-3) lze získat přírůstkový tvar PSD regulátoru [118].

$$\nabla u(kT) = K\left\{\left(e(kT) - e[(k-1)T]\right) + \frac{T}{T_I}e(kT) + \frac{T_D}{T}\left(e(kT) - 2e[(k-1)T] + e[(k-2)T]\right)\right\} \quad (7\text{-}4)$$

7.1.1 Vstupy a výstupy

V implementacích číslicových PID regulátorů jsou obvykle používány proudové nebo napěťové vstupy. Proudové vstupy jsou nejčastěji 4-20 mA, dále 0-20 mA, 10-50 mA. Napěťové vstupy mohou být 0-10 V, 0-5 V, -10-10 V.

Dalšími typy vstupů mohou být např. speciální vstupy pro termočlánky nebo odporové snímače teploty (např. Pt100). Pro optické snímače polohy a rychlosti lze použít impulsní vstupy [120].

K převodu analogových vstupních signálů na číslo jsou podle [119] používány dvanáctibitové, popř. šestnáctibitové A/D převodníky. Nutnou podmínkou správné funkce číslicového regulátoru je splnění vzorkovacího teorému. Tato podmínka bývá obvykle splněna, její splnění však nebývá postačující. Správná volba vzorkovací periody je velmi důležitá, protože může výrazně ovlivnit regulační děj. Podle [118] lze za vhodnou vzorkovací periodu považovat takovou hodnotu, při které nedojde ke zhoršení regulačního děje o více než 15 % oproti použití analogického spojitého regulátoru. Příliš krátká perioda vzorkování vede k velkým akčním zásahům a ke zvyšování nároků na rychlost výpočtů i převodníků a obecně na celý regulační obvod. Při zvětšování periody vzorkování se zvětšuje vliv sumační složky a snižuje vliv složky

diferenční. Při příliš dlouhé vzorkovací periodě dochází k destabilizaci regulačního obvodu.

Výstupy regulátoru mohou být spojité nebo nespojité, podle druhu regulované veličiny. Spojité, tj. analogové výstupy jsou obvykle proudové 4 – 20 mA, případně 0 – 20 mA. Mohou být také napěťové, např. 0 – 10 V. Jako nespojité výstupy jsou často využívány výkonové binární reléové výstupy [120]. Tyto výstupy mohou sloužit pouze jako dvoustavové pro jednoduchou regulaci nebo je lze využít pro pulsně-šířkovou modulaci (PWM) a řídit jimi přímo např. menší stejnosměrné motory.

7.1.2 Výpočet

Vlastní výpočet akčního zásahu regulátoru, popř. jeho přírůstku (viz výše), probíhá přesně podle daných rovnic, převedených na sekvenci jednotlivých mezivýpočtů. Výpočty jsou prováděny s datovými typy s pohyblivou řádovou čárkou (`float`, `double`), které nabízení mnohem větší rozsah a přesnost výpočtů při stejné paměťové náročnosti. Reálné datové typy s sebou při softwarovém řešení přinášejí také nevýhodu v podobě větší časové náročnosti. Moderní mikrokontroléry však disponují dostatečným výkonem i pro tyto výpočty. Výraznou výhodou pro realizaci nejen řídicích algoritmů je pak přítomnost jednotky pro hardwarové výpočty s plovoucí řádovou čárkou (FPU), která umožňuje regulační výpočty mnohonásobně zrychlit.

Musí docházet k cyklickému vykonávání těchto výpočtů s periodou rovnou dané vzorkovací periodě T. To je obvykle zajištěno hardwarovým časovačem, který po každém uplynutí vzorkovací periody vyšle požadavek k přerušení a obslužná rutina tohoto přerušení pak spustí smyčku programu, která provede výpočet.

Velmi důležitým praktickým požadavkem je, že nejdelší možná doba provádění programu (*worst-case execution time*, WCET) musí být kratší než vzorkovací perioda regulátoru. Do této doby je nutné zahrnout nejen dobu vlastního výpočtu akčního zásahu, ale také doby převodu vstupních analogových hodnot na číslo, případnou úpravu těchto hodnot, nastavení výstupu včetně doby převodu binární hodnoty zpět na analogovou a veškeré další operace, související přímo s danou řídicí úlohou. Definujeme-li tuto dobu WCET pro daný regulátor jako C, pak musí platit následující podmínka [121].

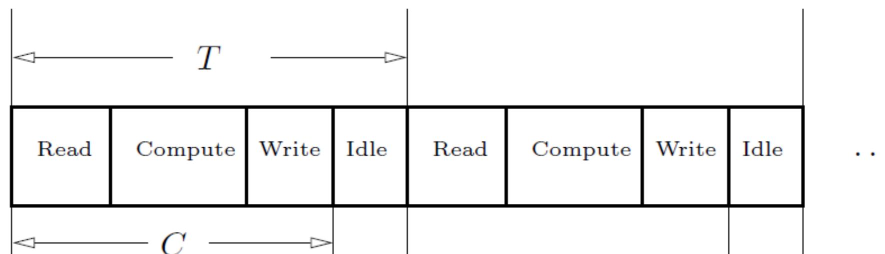
$$C < T \tag{7-5}$$

Tato podmínka je také znázorněna na Obr. 7-1. Do celkové doby výpočtu je zde zahrnuto také čtení vstupů (*Read*) a nastavení výstupů (*Write*). Čas, nevyužitý výpočtem je označen jako *Idle* a lze jej využít např. k obsluze uživatelského rozhraní nebo ke komunikaci s nadřazeným systémem.

Pokud není podmínka (7-5) splněna, není možné zajistit správnou odezvu regulátoru a může dojít k podstatnému zhoršení regulačního pochodu a v některých případech např. i k nebezpečným situacím. Hodnota WCET je závislá na konkrétní

architektuře/platformě a její určení je obtížné zvláště u moderních procesorů, využívajících různé techniky pro zvýšení výkonu [121].

Obr. 7-1: Cyklické provádění řídicího programu s periodou T [121]



7.1.3 Další praktické požadavky

Praktické realizace PSD regulátorů se obvykle doplňují některými funkcemi. Základní algoritmus může být modifikován pro zmenšení překmitu výstupní veličiny při změnách žádané hodnoty. To jsou regulátory S-PD a PS-D [119]. V případě algoritmu PS-D vstupuje regulační odchylka do proporcionální a integrační (sumační) části regulátoru, do derivační (diferenční) části vstupuje pouze záporně vzatá regulovaná veličina. Tím dochází ke zmenšení vlivu rychlých změn žádané hodnoty a tím ke překmitu regulované veličiny. V případě S-PD regulátoru pak záporně vzatá regulovaná veličina vstupuje do derivační i proporcionální části a regulační odchylka vstupuje pouze do sumační části, čímž lze dosáhnout dalšího omezení rychlých změn akčního zásahu regulátoru.

Dalším problémem při praktické realizaci je derivační složka a její náchylnost na šumy a poruchové signály. Ve spojitém regulátoru je díky setrvačnostem zajištěna částečná odolnost proti rychlým změnám regulační odchylky. Číslicová realizace je však na tyto vlivy náchylná, protože, jak bylo uvedeno, výpočet se provádí z navzorkovaných hodnot přesně podle zadaných rovnic. Tím může dojít k příliš velkým změnám akční veličiny. Tomuto jevu lze zabránit filtrací derivační složky např. podle [119].

Velikost reálného akčního zásahu je vždy nějakým způsobem omezena (velikost napájecího napětí, doraz akčního členu). Pokud dojde k takovému omezení a regulační odchylka je dále integrována, zvětšuje se velikost akčního zásahu, aniž by se projevila na výstupu regulátoru. Až se změnou znaménka regulační odchylky začne se začne integrační složka zmenšovat, což vyžaduje určitý čas. Tím vzniká tzv. *wind-up* zpoždění, které zhoršuje kvalitu regulačního děje. U číslicových regulátorů je tento problém opět výraznější, protože zde nejsou žádná fyzikální omezení a jediné praktické omezení je dáno rozsahem reprezentací čísel v počítači/mikrokontroléru. Proto je třeba algoritmicky omezit další načítání integrační složky regulátoru, pokud akční veličina překročí určitou maximální hodnotu, a to až do doby, kdy dojde ke změně znaménka

regulační odchylky [118]. K ochraně proti přeintegrování lze použít různých metod s různými vlastnostmi. Základem všech metod je detekce saturace akční veličiny regulátoru. V [122] jsou uvedeny nejdůležitější metody.

Pro praktické využití spojitých i diskrétních regulátorů se dále používá mechanismus beznárazového přepínání mezi ručním a automatickým režimem. V ručním režimu lze nastavovat akční veličinu. Beznárazové přepínání musí zajistit hladký přechod mezi jednotlivými režimy při nulové regulační odchylce. Při použití polohového algoritmu je nutné zajistit, aby při přepnutí byl výstup regulátoru shodný s ručně nastavenou hodnotou. U přírůstkového algoritmu je pak předchozí hodnota akčního zásahu vždy uložena do paměti bez ohledu na to, ve kterém režimu byla nastavena [118]. Pro beznárazové přepínání lze použít např metodu zpětného sledování signálu, která je vhodná i pro vícerozměrné regulační systémy. Mimo to ji lze využít také k ochraně proti *wind-up* zpoždění s proměnnými hodnotami mezí saturace akční veličiny [122].

7.2 Komplexní regulační systémy

V předcházející části byla obecně popisována implementace regulačního systému s jediným regulátorem. Taková implementace je poměrně snadná, a obvykle nevyžaduje použití zvláštních technik ani operačního systému jako prostředníka mezi regulačním algoritmem a hardwarem použitého počítače nebo mikrokontroléru. Díky tomu je velmi jednoduchá a bezpečná.

V řídicích úlohách se však lze setkat také se složitějšími systémy, které, mají-li být úspěšně řízeny, vyžadují komplexnější regulační obvody. Tyto systémy mohou mít více stupňů volnosti, které je obvykle nutné řídit současně a je tedy nutné použít více současně pracujících regulátorů. Pokud má být takové řízení realizováno počítačem nebo mikrokontrolérem, který je ze své podstaty sekvenčním systémem a neumožňuje tedy paralelní zpracování, je třeba zajistit spolehlivé plánování a provádění jednotlivých operací a výpočtů tak, aby bylo vždy zajištěno, že každý výpočet bude proveden se správnými a aktuálními daty a bude ukončen v daném čase. Toto je základní předpoklad pro správnou funkci systémů reálného času.

7.2.1 Systémy s jedinou vzorkovací periodou

Nejjednodušším případem komplexního řídicího systému je soustava několika regulátorů, pracujících se stejnou periodou vzorkování. To znamená, že během této periody musí být sekvenčně proveden výpočet všech těchto regulátorů, včetně čtení vstupů a nastavení výstupů tak, jak bylo uvedeno výše, aniž by celkový čas překročil dobu vzorkovací periody. Definujeme-li pro systém s n regulátory $c_1 \ldots c_n$ se společnou

vzorkovací periodou T dobu WCET i-tého regulátoru jako C_i, musí být pro realizaci takového systému splněna podmínka

$$\sum_{i=1}^{n} C_i < T\,. \tag{7-6}$$

Podmínka (7-6) je v podstatě rozšířením výše uvedené podmínky (7-5) pro systém s více regulátory. Jestliže je tato podmínka splněna, lze použít jednoduchý plánovací mechanismus a spouštět jednotlivé výpočty sekvenčně, popř. by bylo možné jednotlivým regulátorům přiřadit priority podle důležitosti a spouštět jejich výpočty podle těchto priorit. Tato metoda, kdy jsou programy spouštěny jeden po druhém, umožňuje využít již vytvořený kód pro všechny regulátory a jednoduchými úpravami např. měnit počet regulátorů. Při použití uvedeného plánovacího mechanismu je pak možné např. některý regulátor jednoduše vyřadit z provozu. Metoda nabízí dobrou modularitu, není však příliš efektivní z pohledu využití procesorového času.

Větší efektivity, ovšem s následkem výrazně menší modularity, lze pak dosáhnout spojením všech regulátorů do jediného programu a využitím optimalizačních nástrojů k vygenerování kódu pro celý řídicí systém. Některé části výpočtů jsou společné pro více regulátorů a je tedy možné provést tyto výpočty pouze jednou. Tím dochází k redukci počtu operací a zefektivnění kódu a je možné na daném procesoru dosáhnout kratší doby vzorkování [121].

7.2.2 Systémy s více vzorkovacími periodami

Obecnějším a častějším případem regulovaného systému je soustava s několika stupni volnosti, které mají různou dynamiku. To znamená, že pro řízení takové soustavy je nutné použít systém regulátorů s různými vzorkovacími periodami. V [121] je takový systém s n regulátory popsán jako množina dvojic (c_i, T_i) pro $i = 1 \ldots n$, kde T_i je celočíselná vzorkovací perioda i-tého regulátoru c_i. Pro tento systém lze dále definovat základní periodu, tj. nejkratší periodu, se kterou bude systém pracovat, jako největšího společného dělitele (*greatest common divisor*, gcd) vzorkovacích period jednotlivých regulátorů. Pro základní periodu T tedy platí.

$$T = \mathrm{gcd}(T_1, \ldots, T_n)\,. \tag{7-7}$$

V [121] je definována také tzv. super-perioda P, která představuje dobu, po které se bude opakovat spouštění jednotlivých regulátorů. Pokud tedy bude nalezena vyhovující sekvence spouštění pro jednu super-periodu, lze řídicí systém považovat za funkční. Super-periodu lze definovat jako nejmenší společný násobek vzorkovacích period jednotlivých regulátorů (*least common multiple*, lcm).

$$P = lcm(T_1, \ldots, T_n) \tag{7-8}$$

Přirozeným a nejjednodušším řešením tohoto problému je spouštění úloh jednotlivých regulátorů vždy v násobcích základní periody, odpovídajících jednotlivým

vzorkovacím periodám T_i. Toto řešení je však neefektivní. V některých cyklech je třeba, aby byly provedeny výpočty všech regulátorů, zatímco v ostatních cyklech je prováděno mnohem méně výpočtů. V případě nesoudělných vzorkovacích period je pak možné, že budou některé cykly zcela nevyužity, neuvažujeme-li další funkce, jako např. obsluhu uživatelského rozhraní. Nejkratší základní perioda, dosažitelná s použitím daného procesoru, je určena součtem časů WCET jednotlivých regulátorů. Opět musí být splněna podmínka (7-6).

Za předpokladu, že lze úlohu zpracovávající výpočet daného regulátoru rozdělit na několik částí, které lze provádět v méně využitých cyklech, je možné dosáhnout efektivnějšího využití procesorového času. Podmínkou je, aby byla celá úloha dokončena v odpovídajícím časovém intervalu. Pro první spuštění úlohy regulátoru c_i je to interval $(0;T_i)$. Obecně pro k-té spuštění bude tento interval $([k-1]\cdot T_i;k\cdot T_i)$. Spodní hranice intervalu je v [121] označena jako čas spuštění (*release time*) a horní hranice jako *deadline*, tj. časový limit pro dokončení úlohy. Tento způsob řešení lze provést buď statickým rozdělením při překladu nebo použitím operačního systému reálného času.

Statické dělení

První způsob spočívá ve statickém rozdělení kódu na několik částí s přibližně stejnou délkou provádění a spouštění těchto částí rovnoměrně v daném intervalu. Tím lze dosáhnout lepší efektivity. V [121] je uvedeno, že tato metoda je v praxi populární a je použita v některých komerčních nástrojích.

Použití RTOS

Další možností, jak bylo uvedeno, je využití operačního systému reálného času. Každý regulátor je zde reprezentován samostatnou úlohou a tyto úlohy jsou spouštěny preemptivním plánovačem operačního systému. Úlohy s delší periodou jsou prováděny pouze když je k dispozici dostatek času a jsou přerušovány úlohami, které mají být vykonány dříve. Nevýhodou je určitý čas, strávený přepínáním kontextu při změně běžící úlohy. Z tohoto důvodu je doba přepnutí kontextu jedním z nejdůležitějších parametrů každého operačního systému reálného času.

Jestliže definujeme relativní množství času, strávené vykonáváním i-té úlohy jako $\frac{C_i}{T_i}$, pak pro n úloh bude celkové relativní množství času, strávené jejich zpracováním, tj. v podstatě koeficient využití procesoru (*utilization factor*), dáno následujícím vztahem [123].

$$U=\sum_{i=1}^{n}\frac{C_i}{T_i} \tag{7-9}$$

Jestliže má být plánování realizovatelné, tj. koeficient využití procesoru musí být menší než 100 %, pak musí platit

$$U < 1 \quad \Rightarrow \quad \sum_{i=1}^{n} \frac{C_i}{T_i} < 1 . \tag{7-10}$$

Podmínka (7-10) je uvedena také v [121].

K plánování lze použít různých strategií. Dvě nejpopulárnější plánovací strategie jsou podle [121] EDF (*earliest deadline first*) a RM (*rate-monotonic*).

Metoda EDF

První uvedená metoda spočívá v tom, že je vždy vybrána úloha s nejbližším časem pro ukončení (*deadline*) a tato je prováděna, tj. je jí přidělena nejvyšší priorita. Jedná se tedy o dynamickou změnu priorit úloh v závislosti na časových limitech. Pokud jsou tyto časy stejné u více úloh, je výběr libovolný s tím, že přednost je dána úloze, která již případně běží, aby bylo minimalizováno přepínání kontextu. Tato metoda je podrobněji popsána v [123] jako *Deadline Driven Scheduling*.

Metoda RM

Druhá metoda využívá statické přidělování priorit, které vychází z frekvence provádění jednotlivých úloh, tj. priority jsou úlohám přiděleny podle velikosti vzorkovacích period. Úloha pro regulátor s nejkratší vzorkovací periodou bude mít nejvyšší prioritu, úloha pro regulátor s nejdelší periodou pak bude mít nejnižší prioritu. Tato metoda provádí více preempcí a je tedy méně efektivní než výše uvedená, je však jednodušší ji v operačních systémech implementovat [121].

V [121] je dále uveden příklad systému se třemi regulátory. Tento systém je popsán $S_{123} = \{(c_1,1),(c_2,2),(c_3,3)\}$. Základní perioda je $T = 1$ a super-perioda $P = 6$. Intervaly pro spouštění jednotlivých regulátorů c_1 až c_3 jsou následující.

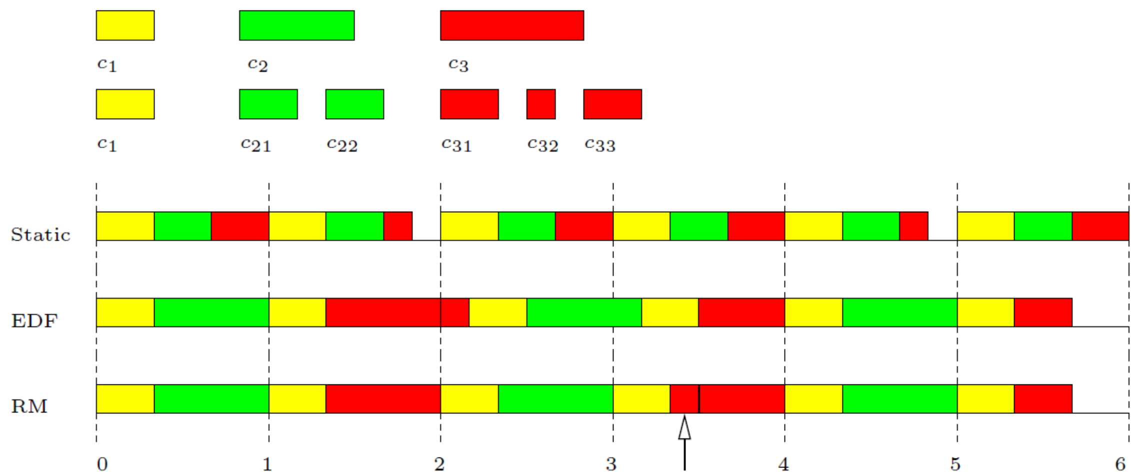
$$\begin{aligned} c_1 &: \quad (0;1),(1;2),(2;3),(3;4),(4;5),(5;6) \\ c_2 &: \quad (0,2),(2,4),(4,6) \\ c_3 &: \quad (0,3),(3,6) \end{aligned}$$

Na Obr. 7-2 je srovnání tří uvedených metod dělení programu, tj. statické dělení (na obrázku označeno Static), algoritmus EDF a algoritmus RM. Statickým rozdělením jsou úlohy regulátorů c_2 a c_3 rozděleny na c_{21}, c_{22} a c_{31}, c_{32}, c_{33} a tyto jednotlivé části jsou spouštěny tak, aby byly dodrženy uvedené časové intervaly.

Na Obr. 7-2 je dále vidět, že při použití metody EDF není první instance úlohy c_3 přerušena třetí instancí úlohy c_1, protože tyto úlohy mají stejný časový limit pro vykonání (*deadline*) a je tedy v souladu s výše uvedeným principem metody zachováno provádění úlohy c_3. K tomuto však nedochází u metody RM, kdy je kvůli statickému

přidělení priorit vykonávání úloh c_2 a c_3 vždy v daném okamžiku přerušeno úlohou c_1. To má za následek, že první instance úlohy c_3 není dokončena v daném intervalu $(0;3)$, jak je patrné na Obr. 7-2. Zbylá část výpočtu (na obrázku označena šipkou) je provedena až po překročení horního limitu tohoto intervalu. Metodu RM tedy nelze pro uvedený systém na dané platformě použít.

Obr. 7-2: Srovnání statického dělení kódu, EDF a RM pro systém S_{123} [121]



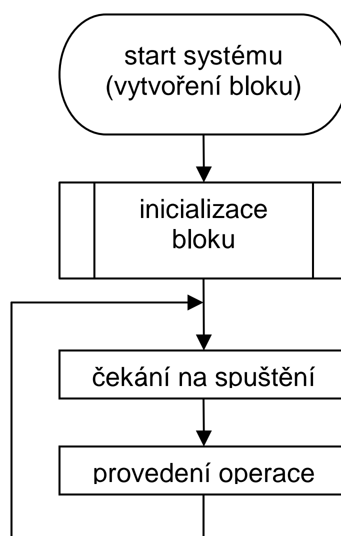
7.3 Návrh PSD regulátoru jako úlohy operačního systému

V klasické smyčkové implementaci se lze na regulátor dívat jako na sekvenci operací a výpočtů, která je opakovaně spouštěna obvykle s využitím hardwarového časovače. Pokud je třeba, aby pracovalo současně více regulátorů, je obvykle nutné tuto sekvenci modifikovat. Při složitějších úpravách programu pak dochází nejen ke zvyšování obtížnosti údržby a ladění, ale i ke snižování celkové spolehlivosti systému a jeho schopnosti vyhovět daným požadavkům na časové chování. Může tedy dojít i k selhání takového systému.

Z pohledu operačního systému je vhodnější o regulátoru uvažovat jako o funkčním bloku, který disponuje určitými vstupy a výstupy. Tento blok je možné spouštět a zastavovat, měnit prioritu jeho provádění, popř. v případě potřeby dynamicky vytvářet další instance tohoto základního bloku. Pokud je daná řídicí úloha rozdělena na takovéto funkční bloky, je tím snížena obtížnost ladění a zvýší se celková přehlednost vytvořeného systému. V případě správného návrhu pak také dochází ke zvýšení spolehlivosti a selhání některé z méně kritických funkcí nemusí způsobit selhání celého systému, což je v některých aplikacích velmi důležité (tzv. *fault-tolerant* systémy).

Principiální funkce popsaného bloku může vypadat např. tak, jak je naznačeno na Obr. 7-3. Po úvodní inicializaci bloku, provedené při spuštění systému, popř. při jeho vytvoření, je jeho vykonávání pozastaveno až do doby, kdy je zapotřebí. Tento blok je pak spouštěn pouze v případě potřeby a tím je umožněn běh dalších bloků. Samozřejmě je možné, že vykonávání bloku bude přerušeno např. z důvodu příchodu požadavku s vyšší prioritou. Správné a rychlé přepnutí kontextu a jeho opětovné obnovení při návratu musí zajistit operační systém.

Obr. 7-3: Principiální znázornění činnosti úlohy operačního systému



Z uvedeného principu vychází vlastní návrh PSD regulátoru jako úlohy pro implementaci s použitím vybraných OS reálného času. V inicializační části po spuštění úlohy jsou vytvořeny všechny potřebné proměnné, vypočteny některé konstantní hodnoty a také inicializovány použité synchronizační prostředky.

Následuje hlavní smyčka úlohy, kde je vždy nejprve proveden požadavek na spuštění A/D převodu. Po tomto požadavku je úloha pozastavena (blokována) a čeká na synchronizační pokyn, že byl převod ukončen. Po odblokování a uložení převedených dat je třeba provést kontrolu, zda nedošlo ke změně žádané hodnoty nebo některého parametru regulátoru. Pokud ano, jsou tyto hodnoty změněny. Následně je převedené binární číslo přepočteno na reálnou hodnotu vstupního signálu a je proveden vlastní regulační výpočet. Po provedení výpočtu je hodnota přepočtena zpět na binární číslo a toto číslo je posláno k převodu na analogovou hodnotu.

8 IMPLEMENTACE

V této kapitole bude popsán postup implementace nejprve obou vybraných operačních systémů od stažení distribuce systému z internetu přes vytvoření a konfiguraci pracovního prostředí až po implementaci navržených regulačních struktur.

Vybrané OS reálného času byly staženy v podobě zdrojových souborů, které bylo nutno připojit k vlastnímu projektu. Programy byly tvořeny v jazyku C. Pro implementaci programového vybavení byly použity vývojové nástroje z projektu ***GNU Tools for ARM Embedded Processors***, které jsou spravovány zaměstnanci společnosti ARM [124]. Tyto volně šiřitelné nástroje jsou odvozeny od běžných nástrojů GNU a jsou určeny pro procesory s jádry Cortex-M a Cortex-R. Je možné je použít jak na systémech s Linuxovým jádrem, tak i na operačních systémech Windows a Mac OS. Mezi nástroji jsou nejen základní programy jako překladač, linker apod., ale také např. ladicí program ***gdb*** nebo programy pro zjištění velikosti přeložených binárních souborů. Pro tvorbu programového vybavení byla použita verze 4.7 2013q2. Po stažení předsestavené verze pro Windows v .zip archivu o velikosti přibližně 100 MB je vhodné nastavit systémovou proměnnou PATH tak, aby obsahovala cestu k binárním souborům všech nástrojů, tj. k podadresáři **bin** v kořenovém adresáři distribuce. Tímto nastavením je pak zajištěno pohodlné volání všech programů bez nutnosti nastavovat cestu v každém projektu.

Pro nahrávání přeloženého binárního souboru do FLASH programové paměti mikrokontroléru byl použit program ***STM32 ST-LINK Utility***, který je pro tento účel volně ke stažení z internetových stánek výrobce použitého mikrokontroléru ST Microelectronics.

V počátcích práce bylo rozhodnuto, že pro testování nebude použito žádné integrované vývojové prostředí (IDE). Důvodem pro toto rozhodnutí byla v první řadě špatná dostupnost kvalitních a volně šiřitelných IDE, popř. jejich složitá konfigurace, a také absence podpory komunikačního rozhraní ST-LINK/V2, kterým disponuje použitá vývojová deska STM32F4-Discovery. Program ***STM32 ST-LINK Utility*** však disponuje rozhraním CLI (*Command-Line Interface*), tj. rozhraním pro použití z příkazového řádku. Proto bylo rozhodnuto, že překlad i programování mikrokontroléru bude provedeno v jednom kroku, voláním z příkazového řádku.

Dalším použitým nástrojem tedy byl program ***GNU make***, který slouží k automatizaci a usnadnění překladu většího množství zdrojových souborů. Tento program je běžnou součástí Linuxových systémů. Vzhledem k tomu, že v systémech Windows není tato funkce přímo dostupná, byla použita varianta programu ***make*** z distribuce vývojových prostředků ***WinAVR*** [125] verze 20100110 (nejnovější verze) pro mikrokontroléry řady AVR firmy Atmel. Funkce této varianty programu ***make*** se neliší od původního programu z projektu GNU.

Pro řízení překladu slouží speciální soubor **Makefile** který obsahuje popis vzájemných závislostí jednotlivých zdrojových souborů a postup překladu, případně další příkazy. Program ***make***, a tedy i překlad a všechny související operace definované v souboru **Makefile**, lze spustit z příkazového řádku stejnojmenným příkazem *make*, případně s dalšími parametry, kterými lze určit požadovanou činnost programu ***make***. Po zavolání tohoto příkazu je automaticky proveden překlad a je možné automaticky ihned po překladu výsledný binární soubor nahrát do paměti FLASH mikrokontroléru zavoláním CLI funkcí programu ***STM32 ST-LINK Utility*** ze souboru **Makefile** tak, jak bylo popsáno výše. K použití tohoto však nedošlo, bylo shledáno, že pro daný účel je dostačující vždy přeložený binární soubor otevřít v grafickém rozhraní programu ***STM32 ST-LINK Utility*** a spustit programování ručně.

Pro vlastní tvorbu a prohlížení zdrojových kódů byl použit volně šiřitelný univerzální editor ***PSPad***, určený pro systémy Microsoft Windows. Tento editor nabízí zvýraznění syntaxe pro množství různých jazyků, samozřejmě včetně C/C++.

Pro ladění lze využít program ***gdb***, který je součástí výše popsaných vývojových nástrojů.

8.1 Vytvoření projektu pro systém FreeRTOS

Pro práci se systémem FreeRTOS byla jako základ použita struktura převzatá z projektu, staženého z [126]. Tento projekt byl použit již při ověřování funkce samotné vývojové desky a použitého mikrokontroléru (bez použití OS). Projekt již obsahuje téměř vše potřebné k vývoji aplikací pro vývojovou desku STM32F4-Discovery včetně standardních knihoven a ovladačů, dodávaných výrobcem ST Microelectronics, a skriptu pro linker.

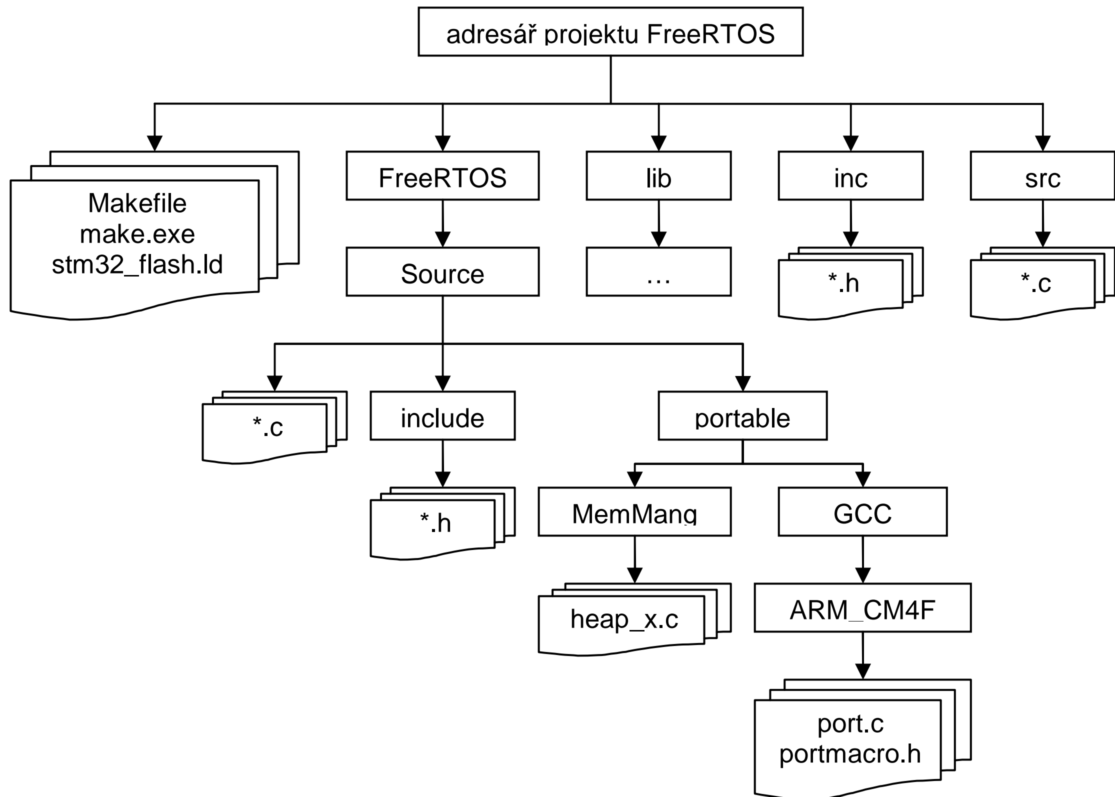
Původní struktura obsahuje tři adresáře. Adresář **src** obsahuje zdrojové soubory aplikace, obslužných rutin přerušení a systémové konfigurace. Adresář **inc** pak podobně obsahuje hlavičkové soubory. Poslední adresář **lib** obsahuje výše zmíněné standardní knihovny a ovladače pro vývojovou desku a vestavěné periferie použitého mikrokontroléru a dále inicializační rutinu vytvořenou v jazyku symbolických adres (**startup_stm32f4xx.s**). Struktura tohoto adresáře je dále dělena obdobně, tj. na zdrojové a hlavičkové soubory a také je provedeno další logické dělení. Knihovny, využívané jádrem mikrokontroléru jsou umístěny odděleně od ovladačů periferií, jejichž použití je volitelné.

Struktura projektu dále obsahuje zmíněný skript pro linker (**stm32_flash.ld**) a dále hlavní soubor **Makefile**, který popisuje vzájemné závislosti jednotlivých zdrojových souborů a postup překladu a sestavení výsledného binárního souboru. Soubor **Makefile** je využíván programem ***make***, který je do adresáře projektu zkopírován.

Úprava pro použití systému FreeRTOS spočívala v první řadě v přidání dalšího adresáře se zdrojovými soubory OS do projektu. Tento adresář byl pojmenován

FreeRTOS a v něm byl vytvořen podadresář **Source** tak, aby byla dodržena struktura adresářů distribuce systému FreeRTOS (viz část 5.2). Do podadresáře **portable** pak byly vloženy pouze zdrojové soubory správy paměti a příslušné soubory portu pro překladač GCC a architekturu ARM Cortex-M4F. Struktura projektu je znázorněna na Obr. 8-1. Pro zachování přehlednosti není znázorněna vnitřní struktura adresáře **lib**.

Obr. 8-1: Zjednodušené znázornění struktury projektu pro systém FreeRTOS



Dále byl upraven hlavní soubor **Makefile**. V tomto souboru jsou definovány všechny zdrojové soubory, které mají být přeloženy (`SRCS`). Je tedy nutné definovat také zdrojové soubory systému FreeRTOS, případně další vytvořené zdrojové soubory.

Tímto je příprava projektu pro použití s operačním systémem FreeRTOS dokončena.

8.2 Vytvoření projektu pro systém ChibiOS/RT

Struktura projektu pro operační systém ChibiOS/RT byla převzata z demonstrační aplikace pro vývojovou desku STM32F4-Discovery, která je součástí distribuce. Při pokusu o první překlad se však objevila chyba, způsobená pravděpodobně nekompatibilitou (demonstrační aplikace jsou určeny pro překlad pod Linuxovými systémy). Po několika neúspěšných pokusech o odstranění této chyby bylo rozhodnuto, že pro práci se systémem ChibiOS/RT bude využit již nainstalovaný balík otevřených

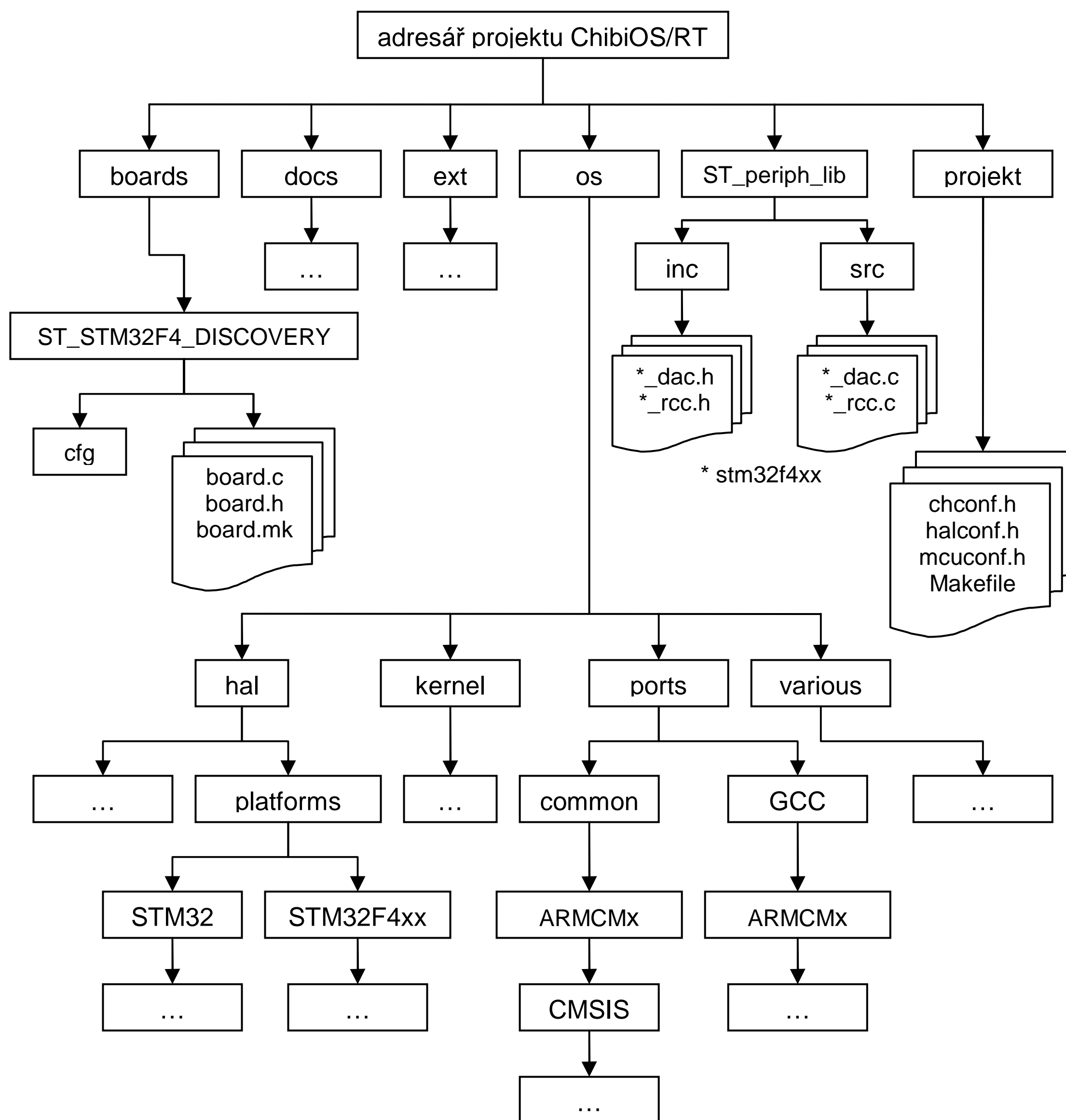
programů ***Cygwin***, který umožňuje emulaci unixových systémů na operačních systémech Microsoft Windows. Byla použita verze 2.819. Více o tomto balíku programů lze nalézt v [127].

Zdrojové soubory projektu je třeba zkopírovat do adresáře, kde je balík ***Cygwin*** nainstalován a to do podadresáře **cyqwin/home/(uživatelské jméno)**. Struktura projektu byla následně upravena následujícím způsobem. Z adresáře **boards** byly vymazány všechny nepotřebné podadresáře pro vývojové desky a ponechán pouze podadresář **ST_STM32F4_DISCOVERY** pro použitou vývojovou desku. Stejně tak byly vymazány všechny podadresáře z adresáře **os/hal/platforms** kromě podadresáře **STM32F4xx** a **STM32** a z adresáře **os/ports** byly ponechány pouze podadresáře **common** a **GCC/ARMCMx**. Následně byly odstraněny adresáře **test** a **testhal**, které slouží k testování funkcí operačního systému a pro projekt nejsou potřebné. Poslední úpravou bylo přesunutí demonstrační aplikace z adresáře **demos/ARMCM4-STM32F407-DISCOVERY** do nově vytvořeného adresáře určeného pro zdrojové soubory projektu **ChibiOS_2.6.3/projekt**. Zbytek demonstračních aplikací byl opět smazán.

Z důvodu absence ovladače pro D/A převodník ve vrstvě HAL systému ChibiOS/RT byl přidán další adresář **ST_periph_lib**, který obsahuje potřebné zdrojové soubory standardních ovladačů periferií od výrobce mikrokontroléru ST Microelectronics.

Bylo také nutné upravit hlavní soubor **Makefile**. První úprava spočívala v nastavení správné cesty ke zdrojovým souborům operačního systému (proměnná `CHIBIOS`). Dále byly přidány zdrojové soubory ovladačů z adresáře **ST_periph_lib** k ostatním zdrojovým souborům v jazyku C (proměnná `CSRC`) a cesta k hlavičkovým souborům těchto ovladačů (proměnná `INCDIR`). Současně byly odebrány soubory pro testování (`TESTSRC` a `TESTINC`) a odebráno vložení souboru **test.mk** pro tyto soubory. Následně bylo také povoleno použití jednotky FPU (`USE_FPU = yes`). Tímto je projekt připraven k použití. Zjednodušené znázornění struktury projektu je na Obr. 8-2.

Obr. 8-2: Zjednodušené znázornění struktury projektu pro systém ChibiOS/RT



8.3 Implementace PSD regulátoru v systému FreeRTOS

Úloha pro PSD regulátor byla nejprve implementována v systému FreeRTOS podle principiálního návrhu uvedeného v části 7.3. Pro synchronizaci úlohy regulátoru s A/D převodníkem byla použita fronta, pomocí které jsou současně posílána převedená data. Další fronta byla použita pro předávání žádané hodnoty, případně parametrů regulátoru, z nadřazené úlohy.

Vzhledem k tomu, že není třeba provádět uvolňování paměti, bylo použita nejjednodušší implementace správy paměti (soubor **heap_1.c**, viz část 5.7). Zásobník byl nastaven na 1000 slov, tj. pro 32-bitové slovo je velikost zásobníku 4000 B.

Takovéto nastavení zásobníku je plně dostačující pro ladění programu se všemi potřebnými proměnnými. Program byl rozdělen do dvou souborů

Ve zdrojovém souboru **main.c** je hlavní funkce `main()`, kde je provedena inicializace vstupně výstupních vývodů mikrokontroléru, nastavení A/D a D/A převodníku, nastavení priorit přerušení a také vytvoření vlastní úlohy pro PSD regulátor. Je zde také obslužná rutina přerušení od A/D převodníku `ADC_IRQHandler()`. V této rutině je pouze otestován příznak ukončení převodu příslušného převodníku a převedená data jsou odeslána do fronty `xADCQueue` pomocí speciální API funkce pro využití v obslužných rutinách přerušení `xQueueSendFromISR()`. Dále je zde provedeno testování, zda odesláním dat do fronty nedošlo k odblokování některé úlohy. Pokud ano, je tato úloha spuštěna ihned po návratu z přerušení.

Druhým souborem je zdrojový soubor **main_tasks.c** a připojený hlavičkový soubor **main_tasks.h**. Ve zdrojovém souboru **main_tasks.c** jsou globálně deklarovány použité fronty a je zde samotná úloha pro PSD regulátor. V hlavičkovém souboru **main_tasks.h** jsou pak uvedeny definice konstant pro regulátor (inicializační časové konstanty a zesílení, rozsah vstupních a výstupních signálů a rozlišení převodníku) a dále globální proměnné.

Úloha pro PSD regulátor je realizována formou nekonečné smyčky. Vývojový diagram úlohy PSD regulátoru je na Obr. 8-3. Po spuštění úlohy je nejprve provedena inicializace. Proměnné pro data z A/D převodníku a pro D/A převodník jsou celočíselného datového typu `integer` o šířce 16 bitů (použité rozlišení je 12 bitů), ostatní proměnné pro regulační odchylku, akční zásah, stav regulátoru a proměnné pro přepočet převedeného čísla na reálnou hodnotu signálu jsou typu `float`, tj. s plovoucí řádovou čárkou s přesností *single precision*, aby bylo možno pro výpočty s nimi využít jednotku FPU.

Po definici a deklaraci potřebných proměnných je vypočítána frekvence spouštění úlohy podle vztahu

$$xFrequency = \frac{1000}{configTICK_RATE_HZ} \cdot T_S\,, \qquad (8\text{-}1)$$

kde $configTICK_RATE_HZ$ je frekvence systémových hodin nastavená v konfiguračním souboru **FreeRTOSConfig.h**

T_S je vzorkovací perioda regulátoru definovaná v souboru **main_tasks.h**.

Následně jsou pomocí API funkce `xQueueCreate()` vytvořeny fronty pro komunikaci úlohy s obslužnou rutinou přerušení, popř. s nadřazenou řídicí úlohou. Po vytvoření front jsou vypočteny reálné hodnoty LSB pro vstupní a výstupní signál.

Následující vztah je uveden pro vstupní signál.

$$fInputLSB = \frac{PSD_IN_RNG}{2^{PSD_IN_RES} - 1}, \qquad (8\text{-}2)$$

kde PSD_IN_RNG je rozsah vstupního signálu (konstanta)
PSD_IN_RES je rozlišení A/D převodníku (konstanta).

Obě konstanty ve vztahu (8-2) jsou definovány v hlavičkovém souboru **main_tasks.h** jako parametry regulátoru. Pro LSB výstupního signálu (proměnná `fOutputLSB`) platí v podstatě stejný vztah, pouze s jinými konstantami, taktéž definovanými v souboru **main_tasks.h**, proto zde nebude uveden. Po tomto výpočtu vstupuje provádění úlohy do hlavní smyčky.

V hlavní smyčce je nejprve zjištěn aktuální systémový čas pomocí API funkce **`xTaskGetTickCount()`** a uložen jako poslední čas spuštění úlohy a následně je spuštěn A/D převod pomocí přímého zápisu do registru CR2 převodníku ADC1, popř. jiného použitého převodníku. Následně je proveden pokus o čtení dat z fronty `xADCQueue`. Pokud data nejsou k dispozici (fronta je prázdná), úloha je zablokována a v tomto stavu čeká až do doby, kdy jsou data dostupná. Jakmile jsou data v pořádku přijata, je proveden přepočet na reálnou hodnotu vstupního signálu podle vztahu

$$fInputVal = PSD_IN_MIN + uiInputData \cdot fInputLSB, \qquad (8\text{-}3)$$

kde PSD_IN_MIN je spodní hranice rozsahu vstupního signálu, definovaná jako konstanta v souboru **main_tasks.h**
$uiInputData$ je binární hodnota z A/D převodníku (`integer` 16 bitů).

Proměnná `fInputVal` tedy představuje vstup do regulátoru, tj. výstup regulované soustavy. Dále je provedena kontrola, zda nejsou data ve frontě `xParQueue`, pomocí které mohou být předávány parametry regulátoru včetně žádané hodnoty. Pro tuto kontrolu je nastaven krátký časový limit. To znamená, že pokud nejsou data v této frontě dostupná během kontroly, je úloha blokována pouze na velmi krátkou dobu a poté pokračuje v činnosti. Tím je zajištěna funkce regulátoru nezávisle na tom, zda byly nebo nebyly změněny jeho parametry.

Následuje vlastní regulační výpočet. Tento výpočet byl vytvořen podle příkladu implementace PSD regulátoru s filtrací derivační složky a dynamickým omezením sumační složky, který je uveden v [128]. Výpočet zde nebude uveden, protože jeho provedení není z hlediska této práce příliš důležité. Lze jej nalézt ve zdrojovém souboru **main_tasks.c** ve funkci **`vPSDTask()`**, popř. v podobné formě v [128].

Po vypočtení akčního zásahu (proměnná `fOutputVal`) je jeho reálná hodnota přepočtena na binární číslo pro převod D/A převodníkem podle vztahu

$$uiOutputData = \frac{fOutputVal}{fOutputLSB}, \tag{8-4}$$

Přepočtená hodnota je následně pomocí funkcí standardních ovladačů periferií od ST Microelectronics odeslána k převodu D/A převodníkem na výstupní analogovou hodnotu akčního zásahu. Poté je úloha zablokována pomocí API funkce **`vTaskDelayUntil()`** a čeká na další spuštění.

8.4 Implementace PSD regulátoru v systému ChibiOS/RT

Implementace úlohy pro PSD regulátor byla provedena také v systému ChibiOS/RT. Tato implementace je z velké části shodná s implementací v systému FreeRTOS, uvedené v předešlé části, a liší se v podstatě pouze z důvodu odlišných funkcí obou systémů. Vývojový diagram na Obr. 8-3 je tedy platný i pro tuto implementaci. Základním rozdílem oproti předešlé implementaci je využití vrstvy HAL, kterou systém ChibiOS/RT nabízí. Konkrétně jde o využití ovladače pro A/D převodník a dále ovladačů pro GPIO vývody mikrokontroléru. Vzhledem k tomu, že vrstva HAL systému ChibiOS/RT nenabízí ovladač pro D/A převodník, bylo nutné k projektu připojit také standardní ovladač pro tento převodník od výrobce mikrokontroléru tak, jak popisuje část 8.2. K předávání paramterů regulátoru byl využit mechanismus asynchronních zpráv, popsaný v části 6.6.6. Velikost zásobníku byla opět nastavena na vysokou hodnotu 4000 B, aby byly při ladění vyloučeny problémy způsobené její příliš malou hodnotou.

Program je opět rozdělen do dvou souborů se stejnými názvy jako u systému FreeRTOS, tj. **main.c** a **main_tasks.c** s hlavičkovým souborem **main_tasks.h**. Ve zdrojovém souboru **main.c** je definována inicializační struktura pro nastavení A/D převodníku `ADCConvGroup`, kde je provedeno nastavení počtu kanálů, spouštění apod. Je zde také odkaz na tzv. *callback* funkci, která je volána po ukončení převodu [129]. Tato funkce je také definována v souboru **main.c** jako **`ADCCallback()`** a pouze nastavuje semafor `ADCSemaphore`, který je využit pro synchronizaci úlohy PSD regulátoru a A/D převodníku.

Ve funkci **`main()`** je nejprve inicializována vrstva HAL (funkce **`halInit()`**). Následuje nastavení D/A převodníku pomocí funkcí standardních ovladačů výrobce ST Microelectronics. Tato inicializace musí následovat až po volání funkci **`halInit()`**, jinak dochází k problémům, které jsou pravděpodobně způsobeny přepisem určitých registrů. Dále jsou nastaveny GPIO vývody mikrokontroléru pomocí funkcí vrstvy HAL systému ChibiOS/RT. Po tomto nastavení již následuje inicializace vlastního

operačního systému pomocí funkce `chSysInit()`. Následně již mohou být spuštěny úlohy.

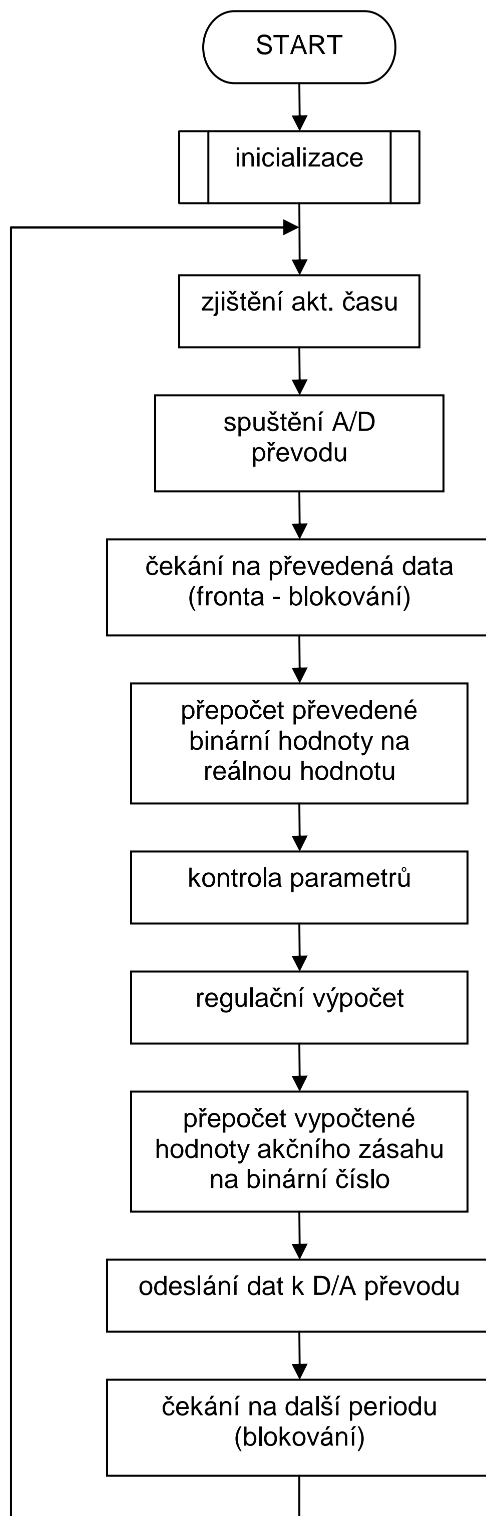
Úlohy a globální proměnné jsou definovány ve zdrojovém souboru **main_tasks.c**, podobně jako u systému FreeRTOS. Úloha pro PSD regulátor je definována jako statická (viz část 6.4). V inicializační části této úlohy jsou opět definovány potřebné proměnné a inicializovány synchronizační prostředky, tj. binární semafor `xADCSemaphore` pro synchronizaci s A/D převodníkem a synchronizační objekt typu *Mailbox* s názvem `xParMailbox` pro případné předávání parametrů z nadřazené úlohy. Je třeba definovat také paměťovou oblast, kterou bude *Mailbox* využívat pro přenos zpráv. Opět jsou provedeny výpočty LSB vstupního a výstupního signálu podle vztahů, uvedených v části 8.3.

Po inicializaci úloha vstupuje do hlavní smyčky, kde je nejprve zjištěn aktuální systémový čas (funkce `chTimeNow()`) a následně je vypočten čas následujícího spuštění pomocí makra `MS2ST()`, které převádí čas v ms na počet inkrementací systémových hodin. Jako parametr tohoto makra lze tedy přímo předat nastavenou hodnotu vzorkovací periody regulátoru v ms.

Dále je spuštěn převod pomocí funkce vrstvy HAL `adcConvert()`. Úloha následně čeká na semaforu `xADCSemaphore` pomocí funkce `chBSemWaitTimeout()` a je blokována, dokud není tento semafor nastaven. Po nastavení semaforu je z připravené globální proměnné vyčtena převedená hodnota. Tato hodnota je přepočtena na reálnou hodnotu vstupního signálu podle vztahu (8-3).

Následuje kontrola změny parametrů pomocí funkce `chMBFetch()` s nastaveným krátkým časovým limitem a vlastní regulační výpočet. Ten je stejný jako v případě implementace v systému FreeRTOS. Stejný je i přepočet akčního zásahu na binární hodnotu (vztah (8-4)). Odeslání této hodnoty k převodu D/A převodníkem na analogovou hodnotu je také shodné s implementací v systému FreeRTOS. Poté je úloha pozastavena pomocí funkce `chThdSleepUntil()` až do příchodu další vzorkovací periody.

Obr. 8-3: Vývojový diagram úlohy pro PSD regulátor



9 VYHODNOCENÍ VLASTNOSTÍ VYBRANÝCH OS REÁLNÉHO ČASU

Posledním bodem zadání je vyhodnocení vhodnosti vybraných operačních systémů reálného času pro realizaci *embedded* systémů zahrnujících regulační algoritmy. Toto vyhodnocení je možné provést z hlediska nabízených funkcí, vlastností a parametrů, uvedených v dokumentaci vybraných operačních systémů, a také z hlediska praktické implementace těchto OS, tj. složitost vlastní implementace, vyžadované nástroje pro vývoj, použitelnost API rozhraní apod. Zde uvedené vyhodnocení bere v úvahu obě tato hlediska. Před vlastním vyhodnocením budou popsány důležité požadavky na regulační systémy využívající OS reálného času.

Operační systém pro použití v regulačních systémech by měl disponovat deterministickým preemptivním plánovačem úloh se systémem nastavení priorit včetně možnosti měnit prioritu úloh za běhu systému. Mělo by být také umožněno nastavit několika úlohám stejnou prioritu a tyto úlohy spouštět po nastavitelných časových intervalech (Round-Robin). Přítomnost dalších funkcí jako dynamická tvorba a mazání úloh během činnosti již není nutná a ve většině regulačních systémů by tyto funkce byly pravděpodobně zbytečné. Velmi vhodná je pak přítomnost podpory hardwarových výpočtů s plovoucí řádovou čárkou, tj. podpora jednotky FPU. Při použití této jednotky lze mnohonásobně zvýšit rychlost regulačních výpočtů.

OS by měl nabízet různé synchronizační a komunikační prostředky, které jsou velmi důležité pro zajištění správné činnosti řídicího systému. Z těchto prostředků jsou pro řídicí systémy důležité zejména semafory a fronty pro předávání zpráv a dat. Semafory poskytují rychlou synchronizaci mezi úlohami a také mezi úlohami a obslužnými rutinami přerušení, fronty lze výhodně použít pro předávání dat např. při čtení převedené hodnoty z A/D převodníku v příslušné obslužné rutině.

Velmi důležitým parametrem operačního systému reálného času je doba přepnutí kontextu. Tato doba spoluurčuje efektivitu celého systému.

Pro realizaci *embedded* systémů je obecně velmi důležitým parametrem OS jeho paměťová náročnost. *Embedded* systémy z důvodu požadavku na nízkou cenu a malé rozměry obvykle disponují omezeným množstvím paměti a to jak programové, tak datové paměti RAM. Je proto vhodné, aby použitý OS reálného času využíval tyto paměti v nejmenší možné míře. S tím je spojena také konfigurovatelnost systému, tj. možnost určení, které z nabízených funkcí budou ve výsledku zahrnuty a které nikoliv.

Z hlediska implementace je vhodné, aby byla k použitému operačnímu systému dostupná kvalitní a přehledná dokumentace, což není zdaleka vždy splněno. Kvalitní a rychlou technickou podporu lze očekávat pouze u komerčních systémů, avšak její přítomnost alespoň v podobě diskusního fóra na internetu bývá přítomna i u volně šiřitelných OS. Dalším požadavkem je přehledný a dobře okomentovaný zdrojový kód.

9.1 Vyhodnocení vlastností systému FreeRTOS

FreeRTOS je pravděpodobně nejpoužívanějším volně šiřitelným operačním systémem reálného času. Je šířen pod upravenou licencí GNU GPL. Díky této úpravě umožňuje použití i v komerčních aplikacích bez nutnosti zveřejnit zdrojový kód za splnění jistých podmínek.

Nabízí preemptivní i kooperativní plánování úloh a koprogramy, neklade omezení týkající se počtu úrovní priorit ani počtu úloh se stejnou prioritou. Úlohy se stejnou prioritou jsou spouštěny pomocí algoritmu Round-Robin, jehož časové kvantum však nelze nastavit, resp. je shodné s dobou periody systémových hodin. To lze považovat za nedostatek. Priority úlohám lze dynamicky měnit a to jak právě běžící úloze, tak i úlohám, které aktuálně neběží. Podobně je možné úlohy také pozastavovat a blokovat. K dispozici je také množství dalších užitečných funkcí. Plánovač úloh je možné pozastavit. Z pohledu správy a plánování úloh je tedy systém FreeRTOS velmi dobře vybavený a poskytuje dostatek funkcí.

Základním synchronizačním prostředkem systému FreeRTOS jsou fronty. Data jsou frontami přenášena pomocí kopií, což je intuitivní a flexibilní. Je možné pomocí front předávat také ukazatele na předávaná data a zvýšit tak efektivitu při předávání většího objemu dat. Použití front je velmi jednoduché. Na mechanismu front jsou pak založeny ostatní synchronizační prostředky, tj. čítací a binární semafory a mutexy. Tyto synchronizační prostředky, zejména fronty, jsou dle mého názoru pro použití v *embedded* regulačních systémech dostatečné. Vhodná je i přítomnost funkcí softwarových časovačů. Systém dále nabízí příznaky událostí.

Součástí distribuce systému jsou také čtyři různé implementace správy paměti, od jednoduché, která nabízí pouze alokaci bez možnosti uvolňování paměti, po složitější správu paměti, která však již nezaručuje deterministické chování. Je tedy možné zvolit takovou implementaci, která vyhovuje požadavkům aplikace. Systém nabízí také podporu jednotky MPU pro mikrokontroléry s jádrem ARM Cortex-M3. Pro většinu regulačních systémů by měla být dostačující nejjednodušší implementace a ani podpora MPU není z tohoto pohledu příliš důležitá.

Doba přepnutí kontextu je v [38] udávána 84 strojových cyklů při použití mikrokontroléru s jádrem ARM Cortex-M3 a překladače Keil a s plnou optimalizací pro rychlost. Při použití vyšších taktovacích frekvencí použitého mikrokontroléru by se tedy doba přepnutí kontextu měla pohybovat nejvýše v jednotkách μs, což je, vzhledem k běžným vzorkovacím periodám v řádu desítek až stovek ms, pro většinu regulačních systémů naprosto dostačující. V [38] je dále uvedena spotřeba paměti jádrem systému 236 B paměti RAM a 5 až 10 kB programové paměti pro architekturu ARM7. Tyto údaje jsou rovněž velmi příznivé pro realizaci *embedded* systémů.

Operační systém FreeRTOS nabízí také poměrně široké možnosti konfigurace. Lze konfigurovat jak základní funkce systému (frekvence hodin, počet úrovní priorit,

minimální velikost zásobníku), tak použití jednotlivých synchronizačních funkcí, koprogramů, časovačů a dalších funkcí. Systém tak lze nakonfigurovat podle požadavků aplikace, což je velmi vhodné.

Dokumentace k systému FreeRTOS je k dispozici zdarma na internetových stránkách projektu, její přehlednost však není příliš vysoká a některé její části jsou zastaralé. Další možností je zakoupení dokumentace formou příručky. Samotný zdrojový kód systému je otevřený a dostupný, dobře komentovaný a poměrně přehledný. Funkce API rozhraní jsou logicky pojmenovány a dodržují jednotnou konvenci. Implementace systému je poměrně snadná a v podstatě veškeré problémy, které se během implementace objevily, byly způsobeny jinými faktory.

Za velmi důležitou vlastnost pro realizaci regulačních systémů lze považovat podporu jednotky FPU pro mikrokontroléry s jádrem ARM Cortex-M4F. Výhodou může být také podpora velkého množství procesorových architektur. V současné době je podporováno více než 30 architektur od téměř dvaceti různých výrobců.

Pro ladění poskytuje systém FreeRTOS sadu trasovacích maker, která jsou volána při určitých systémových událostech, což poskytuje účinný ladicí nástroj.

Je také vhodné poznamenat, že systém FreeRTOS není operačním systémem v pravém smyslu, protože nabízí v podstatě pouze funkcionalitu jádra a žádně další funkce jako např. vrstvu HAL nebo vestavěný souborový systém. Z pohledu implementace regulačních *embedded* systémů je však tato vlastnost spíše výhodou, protože většina z těchto funkcí by s největší pravděpodobností nebyla použita a pouze by se snižovala přehlednost a jednoduchost systému.

Závěrem je možné prohlásit, že FreeRTOS je kvalitní a úsporný operační systém reálného času, vhodný z hlediska funkcí i implementace pro použití nejen v regulačních systémech, ale obecně v mnoha oblastech *embedded* systémů.

9.2 Vyhodnocení vlastností systému ChibiOS/RT

Operační systém reálného času ChibiOS/RT je šířen opět pod upravenou licencí GNU GPL s výjimkou, která za určitých podmínek, v této výjimce uvedených, osvobozuje od nutnosti zveřejnit vytvořený zdrojový kód.

Systém ChibiOS/RT nabízí preemptivní plánovač úloh. Počet úrovní priorit je omezen na 128, toto omezení by však ve většině *embedded* systémů nemělo přinášet žádné problémy. Je povoleno nastavení stejné priority více úlohám, úlohy se stejnou prioritou jsou pak spouštěny algoritmem Round-Robin s nastavitelným časovým kvantem. Tato vlastnost je výhodná, protože umožňuje větší flexibilitu spouštění. Pokud je časové kvantum nastaveno nulové, lze docílit kooperativního módu. Za běhu je možné měnit prioritu pouze právě běžící úloze. Aktuálně běžící úlohu je možné pozastavit popř. ukončit, lze také vyslat požadavek na ukončení jiné úlohy. Systém nabízí dva typy úloh – úlohy statické, které jsou vytvořeni při kompilaci, a úlohy

dynamické, které mohou být tvořeny za běhu systému pomocí některého z alokačních mechanismů. Pro správu úloh a plánování poskytuje systém ChibiOS/RT dostatek funkcí.

Pro synchronizaci a komunikaci nabízí operační systém ChibiOS/RT poměrně velké množství různých prostředků. Mezi tyto prostředky patří čítací a binární semafory, mutexy, příznaky událostí, synchronní a asynchronní zprávy, podmíněné proměnné a fronty. Pro většinu *embedded* systémů včetně systémů regulačních jsou jistě nabízené prostředky i jejich funkce více než dostatečné. Dále jsou nabízeny i funkce softwarových časovačů.

Jádro systému využívá statickou alokaci paměti. Statická alokace je bezpečná a deterministická a pro použití v regulačních systémech je tento způsob dostačující. Je však také možné využít volitelné mechanismy správy paměti, které umožňují dynamickou alokaci v haldě nebo pomocí mechanismu paměťových bloků o pevné velikosti.

Systém ChibiOS/RT je deklarován jako velmi rychlý a kompaktní. Doba přepnutí kontextu je při použití mikrokontroléru z rodiny STM32F4, taktovaném na 168 MHz a při použití překladače GCC je v [83] uvedena 0,40 µs. Pro ostatní architektury ARM se doba přepnutí kontextu pohybuje již v jednotkách µs a nijak výrazně tedy nepřekonává konkurenci. Nicméně je tato doba dostatečná i pro realizaci rychlých regulačních systémů. Velikost programové paměti požadovaná jádrem je podle [83] poměrně nízká a pohybuje od 5 do 10 kB, což je pro realizaci *embedded* systémů výhodné.

Systém zahrnuje kromě vlastního jádra také další subsystémy, zejména vrstvu HAL, která poskytuje ovladače pro množství standardních zařízení pro všechny podporované architektury, dále podporu vývojových desek včetně jejich inicializace a také možnost integrace např. souborových systémů. Přítomnost vrstvy HAL je výhodná, během implementace se však ukázalo, že tato vrstva neobsahuje ovladač pro D/A převodník, což je pro realizaci regulačních systémů podstatný nedostatek. Dodatečné doplnění funkcí pro D/A převodník ze standardních ovladačů ST Microelectronics s sebou přineslo komplikace a snížení přehlednosti kódu.

Jsou nabízeny rozsáhlé možnosti konfigurace. Samozřejmostí je konfigurace frekvence systémových hodin, dále systém umožňuje nastavení časového kvanta pro algoritmus Round-Robin, velikost paměti RAM přidělené systému, optimalizace pro rychlost, atd. Dále je možné vybírat jednotlivé synchronizační a ladicí prostředky, vybírat které ovladače vrstvy HAL budou použity včetně nastavení některých funkcí těchto ovladačů. Je také možno konfigurovat použití různých periferií, specifických pro konkrétní architekturu. Možnosti konfigurace systému jsou tedy výborné.

Dokumentace je zdarma dostupná na internetových stránkách formou strukturovaného dokumentu, který je provázán s aktuálním zdrojovým kódem systému. Tato dokumentace je kvalitní a poměrně přehledná. Na internetových stránkách projektu

je také řada článků týkajících se architektury systému, ale také obsahujících obecné informace z oblasti operačních systémů reálného času. Pro technickou podporu je k dispozici udržované diskusní fórum, kde lze nalézt velmi užitečné informace. Zdrojový kód je otevřený a dobře komentovaný a poměrně přehledný. Názvy funkcí jsou kvůli použití předpony „`ch`“ a také kvůli zkracování v některých případech nepřehledné. Na druhou stranu jsou díky použití předpony jasně odděleny funkce systému od uživatelských funkcí.

K dispozici jsou rovněž ladicí funkce. Jsou to funkce pro trasování, kontrolu parametrů a hlídání stavu systému. Je také k dispozici registr všech aktivních úloh v systému, který lze použít nejen pro ladicí účely.

Velkou výhodou pro realizaci embedded systémů zahrnujících regulační výpočty je podpora jednotky FPU pro architekturu ARM Cortex-M4F. Naopak nevýhodou je podpora jen omezeného množství procesorových architektur. V současné době je podporováno pouze osm různých architektur.

Operační systém ChibiOS/RT je tedy zajímavou alternativou k ostatním OS reálného času. Je rychlý a kompaktní a také nabízí některé přídavné funkce, zejména vrstvu HAL, která může velmi usnadnit vývoj *embedded* systémů. Pro aplikace, které tuto vrstvu využijí, jde o vhodnou volbu. Absence ovladače pro D/A převodník však realizaci regulačního systému poněkud komplikuje a jistě je možné pro tyto aplikace nalézt vhodnější systém.

10 ZÁVĚR

Tato diplomová práce se zabývala výběrem a implementací operačních systémů reálného času do výkonného mikrokontroléru s jádrem ARM Cortex-M4F a následnou implementací PSD regulátorů s využitím těchto operačních systémů.

Nejprve bylo provedeno seznámení s architekturou použitého mikrokontroléru STM32F407VGT6 výrobce ST Microelectronics. V krátkosti byla popsána architektura ARM a jádro Cortex-M4F, dále byla popsána systémová architektura použitého mikrokontroléru a také byly zdokumentovány důležité funkce a možnosti jeho vestavěných A/D a D/A převodníků.

Byl proveden průzkum operačních systémů reálného času, které nabízejí podporu mikrokontrolérů z rodiny STM32 s jádrem ARM Cortex-M4F. Během tohoto průzkumu bylo při bližším zkoumání vlastností některých nalezených operačních systémů zjištěno, že je podporováno pouze jádro Cortex-M4, tj. varianta bez hardwarové jednotky FPU. Vzhledem ke skutečnosti, že pro regulační systémy jsou jedním ze základních požadavků rychlé výpočty s plovoucí řádovou čárkou, byl proto při vyhledávání operačních systémů kladen důraz na přímou podporu jednotky FPU. Tomuto požadavku vyhovělo šest systémů, které byly v práci uvedeny. Zde je třeba poznamenat, že nebylo snahou vyhledat všechny systémy, vyhovující uvedenému požadavku, ale pouze nalézt několik operačních systémů s různými vlastnostmi.

Pro další výběr byly v souladu se zadáním práce upřednostněny volně šiřitelné operační systémy. Následně byly zkoumány základní vlastnosti těchto operačních systémů a byla vyvinuta snaha tyto vlastnosti alespoň stručně zdokumentovat. Na základě tohoto průzkumu pak bylo možné učinit další rozhodnutí ohledně výběru operačních systémů pro implementaci.

Pro implementaci do použitého mikrokontroléru STM32F407VGT6 a následnou implementaci PSD regulátorů byly vybrány systémy FreeRTOS a ChibiOS/RT. Operační systém FreeRTOS byl vybrán jako první, protože je profesionálně vyvíjený a ověřený a nebyly tedy očekávány žádné větší komplikace při jeho implementaci. Jako druhý operační systém byl vybrán systém ChibiOS/RT a to zejména díky nabízeným vlastnostem (krátká doba přepnutí kontextu, vrstva HAL) a dále díky volně dostupné kvalitní a přehledné dokumentaci. Po implementaci a otestování (zejména testování funkčnosti jednotky FPU) vybraných operačních systémů následovala vlastní implementace úloh pro PSD regulátory.

Implementace PSD regulátoru v systému FreeRTOS byla podle očekávání téměř bezproblémová, jediný závažnější problém představovalo chybné nastavení priority přerušení A/D převodníku. Řešení tohoto problému bylo časově poměrně náročné, nicméně úspěšné.

Implementace PSD regulátoru v systému ChibiOS/RT pak přinesla problém v podobě absence ovladače pro D/A převodník ve vrstvě HAL. Proto byly využity

standardní ovladače od výrobce mikrokontroléru ST Microelectronics. Jejich integrace do systému ChibiOS/RT byla poměrně snadná, avšak měla za následek jisté snížení přehlednosti a konzistence kódu.

Obě implementace PSD regulátoru byly tedy úspěšně realizovány. Tyto implementace dovolují použití několika nezávislých regulátorů, v případě, že každý regulátor má přiděleny vlastní převodníky. Žádanou hodnotu, stejně jako nastavitelné parametry regulátorů je možné předávat z nadřazené úlohy.

Jako rozšíření se nabízí implementace kompletního regulátoru, tj. včetně uživatelského rozhraní s možností přepnout do režimu ručního ovládání a také realizace beznárazového přepínání mezi ručním a automatickým režimem.

Stávající implementaci by dále bylo možno doplnit o řídicí úlohu, která by spouštěla jednotlivé regulátory v daných časových okamžicích a dosáhnout tak realizace komplexního řídicího systému pro systém s několika stupni volnosti.

Nakonec bylo provedeno vyhodnocení vlastností vybraných operačních systémů reálného času podle posledního bodu zadání. Nejprve byly uvedeny vlastnosti, které lze z hlediska realizace regulačních *embedded* systémů očekávat, popř. považovat za výhodné. Následně byly vlastnosti obou vybraných systémů srovnány s těmito požadavky. Při vyhodnocení byly kromě těchto vlastností brány v úvahu také další aspekty, jako je kvalita dokumentace a složitost vlastní implementace.

Seznam literatury

[1] ARM. *ARM Architecture Reference Manual* [online]. Copyright © 1996-1998, 2000, 2004, 2005 ARM Limited, Issue I, July 2005 [cit 2013-12-20]. Dostupné z: https://www.scss.tcd.ie/John.Waldron/3d1/arm_arm.pdf

[2] TIŠNOVSKÝ, Pavel. Pohled programátora na mikroprocesory ARM. In: *ROOT.CZ* [online]. 13. 3. 2012 [cit 2013-12-22]. Dostupné z: http://www.root.cz/clanky/pohled-programatora-na-mikroprocesory-arm/

[3] ARM. *ARM Architecture Overview* [online]. [cit 2013-12-25]. Dostupné z: http://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARM_Architecture_Overview.pdf

[4] TIŠNOVSKÝ, Pavel. Mikroprocesory s architekturou ARM. In: *ROOT.CZ* [online]. 6. 3. 2012 [cit 2013-12-22]. Dostupné z: http://www.root.cz/clanky/mikroprocesory-s-architekturou-arm/

[5] TIŠNOVSKÝ, Pavel. Instrukční sada Thumb-2 u mikroprocesorů ARM. In: *ROOT.CZ* [online]. 10. 4. 2012 [cit 2013-12-22]. Dostupné z: http://www.root.cz/clanky/instrukcni-sada-thumb-2-u-mikroprocesoru-arm/

[6] ARM. *ARM® v7-M Architecture Reference Manual* [online]. Copyright © 2006-2010 ARM Limited, Issue C_errata_v3, February 2010 [cit 2013-12-26]. Dostupné z: http://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf

[7] ARM. *ARMv7-M Architecture Application Level Reference Manual* [online]. Copyright © 2006-2007 ARM Limited, Issue B, August 2007 [cit 2013-12-26]. Dostupné z: http://www.pjrc.com/arm/pdf/doc/DDI0405B_arm_v7m_architecture_app_level_reference_manual.pdf

[8] TIŠNOVSKÝ, Pavel. Norma IEEE 754 a příbuzní: formáty plovoucí řádové tečky. In: *ROOT.CZ* [online]. 31. 5. 2006 [cit 2013-12-30]. Dostupné z: http://www.root.cz/clanky/norma-ieee-754-a-pribuzni-formaty-plovouci-radove-tecky/

[9] ARM. *ARM® Cortex™-M4 Processor: Technical Reference Manual, Revision r0p1* [online]. Copyright © 2009, 2010, 2013 ARM Limited, Issue D, 11 June 2013 [cit 2013-12-22]. Dostupné z: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf

[10] DOULOS. *Getting started with CMSIS* [online]. Copyright © 2009 by Doulos [cit 2013-12-23]. Dostupné z: http://www.doulos.com/knowhow/arm/CMSIS/

[11] ARM. CMSIS – Cortex Microcontroller Software Interface Standard. *ARM.com* [online]. © ARM Ltd. Copyright 2013 [cit 2013-12-23]. Dostupné z: http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

[12] ST MICROELECTRONICS. *STM32F405xx, STM32F407xx* [online datový list]. © 2013 STMicroelectronics, Rev 4, June 2013 [cit 2013-12-23]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037051.pdf

[13] ST MICROELECTRONICS. *RM0090 Reference manual* [online]. © 2013 STMicroelectronics, Rev 5, September 2013 [cit 2013-12-23]. Dostupné z: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf

[14] ST MICROELECTRONICS. *UM1472 User Manual: STM32F4DISCOVERY STM32F4 high-performance discovery board.* © 2012 ST Microelectronics, Rev 2, January 2012 [cit 2013-12-30].

[15] ST MICROELECTRONICS. *LIS302DL* [online datový list]. © 2008 STMicroelectronics, Rev 4, October 2008 [cit 2013-12-30]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00135460.pdf

[16] ST MICROELECTRONICS. *MP45DT02* [online datový list]. © 2012 STMicroelectronisc, Rev 5, July 2012 [cit 2013-12-30]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00025467.pdf

[17] CIRRUS LOGIC. *CS43L22* [online datový list]. Copyright © Cirrus Logic, Inc. 2010, Revision F2, MARCH 2010 [cit 2013-12-30]. Dostupné z: http://www.cirrus.com/en/pubs/proDatasheet/CS43L22_F2.pdf

[18] KUČERA, Pavel. *Operační systémy reálného času: Úvod* [online prezentace]. MRTS/NRTS 2013-2014 [cit 2014-1-5]. Dostupné z: http://taceo.eu/mrts.php

[19] DI SIRIO, Giovanni. RTOS Concepts. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-1-5]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:rtos_concepts

[20] REAL TIME ENGINEERS. FreeRTOS. *Freertos.org* [online]. [cit 14-5-15]. Dostupné z: http://www.freertos.org/index.html

[21] NUTTX. NuttX Real-Time Operating System. *NuttX Real-Time Operating System* [online]. [cit 2014-5-15]. Dostupné z: http://nuttx.org/

[22] MICRIUM. μC/OS-II Overview. *Micrium* [online]. [cit 2014-5-15]. Dostupné z: http://micrium.com/rtos/ucosii/overview/

[23] MICRO DIGITAL. SMX® RTOS. *Micro digital* [online]. [cit 2014-5-15]. Dostupné z: http://www.smxrtos.com/rtos/

[24] DI SIRIO, Giovanni. ChibiOS/RT Homepage. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-5-15]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php

[25] ECOS. Home Page. *eCos* [online]. [cit 2014-5-15]. Dostupné z: http://ecos.sourceware.org/

[26] NUTTX. About. *NuttX Real-Time Operating System* [online]. [cit 2014-5-15]. Dostupné z: http://nuttx.org/doku.php?id=documentation:about

[27] ARPACI-DUSSEAU, Remzi H., Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces: 7 Scheduling: Introduction* [online]. [cit 2014-5-15]. Dostupné z: http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

[28] NUTTX. Supported Platforms. *NuttX Real-Time Operating System* [online]. [cit 2014-5-15]. Dostupné z: http://nuttx.org/Documentation/NuttX.html#platforms

[29] MICRIUM. μC/OS-II Specifications. *Micrium* [online]. [cit 2014-5-15]. Dostupné z: http://micrium.com/rtos/ucosii/specifications/

[30] MICRIUM. μC/OS-II Ports. *Micrium* [online]. [cit 2014-5-15]. Dostupné z: http://micrium.com/rtos/ucosii/ports/

[31] MICRIUM. μC/OS-II Documentation Home. *Micrium Doc* [online]. [cit 2014-5-15]. Dostupné z: https://doc.micrium.com/pages/viewpage.action?pageId=10753158

[32] ECOS. About eCos. *eCos* [online]. [cit 2014-5-15]. Dostupné z: http://ecos.sourceware.org/about.html

[33] MASSA, Anthony J. *Embedded Software Development with eCos*™ [online]. Upper Saddle River: Prentice Hall Professional Technical Reference, 2003. [cit 2014-5-15]. Dostupné z: http://ptgmedia.pearsoncmg.com/imprint_downloads/informit/perens/0130354732.pdf

[34] ITRON. Micro-ITRON4.0 Specification. *ITRON* [online]. [cit 2014-5-15]. Dostupné z: http://www.ertl.jp/ITRON/SPEC/mitron4-e.html

[35] ECOS. SMP Support. *eCos Reference Manual* [online]. [cit 2014-5-15]. Dostupné z: http://ecos.sourceware.org/docs-3.0/ref/kernel-smp.html

[36] ECOS. eCos Reference Manual. *eCos Reference Manual* [online]. [cit 2014-5-15]. Dostupné z: http://ecos.sourceware.org/docs-3.0/ref/ecos-ref.html

[37] REAL TIME ENGINEERS. About FreeRTOS. *Freertos.org* [online]. [cit 2014-3-11]. Dostupné z: http://www.freertos.org/RTOS.html

[38] REAL TIME ENGINEERS. FreeRTOS FAQ – Memory Usage, Boot Times & Context Switch Times. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/FAQMem.html

[39] REAL TIME ENGINEERS. Features Overview. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/FreeRTOS_Features.html

[40] REAL TIME ENGINEERS. Software Timers. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/RTOS-software-timer.html

[41] REAL TIME ENGINEERS. Event Bits (or flags) and Event Groups. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/FreeRTOS-Event-Groups.html

[42] REAL TIME ENGINEERS. Trace Hook Macros. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/rtos-trace-macros.html

[43] REAL TIME ENGINEERS. Stack Usage and Stack Overflow Checking. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/Stacks-and-stack-overflow-checking.html

[44] REAL TIME ENGINEERS. Android, FreeRTOS top EE Times' 2013 embedded.survey. *EE Times* [online]. [cit 2014-3-14]. Dostupné z: http://www.eetimes.com/document.asp?doc_id=1263083

[45] REAL TIME ENGINEERS. License Details. *Freertos.org* [online]. [cit 2014-3-14]. Dostupné z: http://www.freertos.org/a00114.html

[46] HIGH INTEGRITY SYSTEMS. SAFERTOS, the first RTOS pre-certified to IEC 61508. *SAFERTOS* [online]. [cit 2014-3-14]. Dostupné z: http://www.highintegritysystems.com/safertos/

[47] HIGH INTEGRITY SYSTEMS. Certification and Standards of our Safety Critical RTOS, SAFERTOS. *SAFERTOS* [online]. [cit 2014-3-14]. Dostupné z: http://www.highintegritysystems.com/safertos/certification-and-standards/

[48] REAL TIME ENGINEERS. Official FreeRTOS Ports. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/RTOS_ports.html

[49] REAL TIME ENGINEERS. FreeRTOS Windows Simulator. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/FreeRTOS-Windows-Simulator-Emulator-for-Visual-Studio-and-Eclipse-MingW.html

[50] REAL TIME ENGINEERS. FreeRTOS Ports. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/a00090.html

[51] REAL TIME ENGINEERS. RTOS Source Code Download Instructions. *Freertos.org* [online].[cit 2014-3-28]. Dostupné z: http://www.freertos.org/a00104.html

[52] REAL TIME ENGINEERS. FreeRTOS+ Ecosystem Showcase. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/FreeRTOS-Plus/index.shtml

[53] REAL TIME ENGINEERS. Source Organization. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/a00017.html

[54] REAL TIME ENGINEERS. Customisation. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/a00110.html

[55] GOYETTE, Rich. *An Analysis and Description of the Inner Workings of the FreeRTOS Kernel* [online]. 1 April 07 [cit 2014-3-16]. Dostupné z: http://www.mikrocontroller.net/attachment/95930/FreeRTOSPaper.pdf

[56] SOURCEFORGE. FreeRTOS Real Time Kernel (RTOS). *SourceForge.net* [online]. [cit 2014-3-16]. Dostupné z: http://sourceforge.net/projects/freertos/files/FreeRTOS/

[57] D'SOUZA Deepak. *Free RTOS Implementation* [online prezentace]. 23 August 2011 [cit 2014-3-16]. Dostupné z: http://drona.csa.iisc.ernet.in/~deepakd/ukieri/ictac-school/RTOS-implementation.pdf

[58] REAL TIME ENGINEERS. History.txt. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/History.txt

[59] REAL TIME ENGINEERS. Tasks. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/RTOS-task-states.html

[60] MELOT, Nicolas. *Study of an operating system: FreeRTOS* [online]. [cit 2014-3-16]. Dostupné z: http://wiki.csie.ncku.edu.tw/embedded/FreeRTOS_Melot.pdf

[61] REAL TIME ENGINEERS. vTaskSetApplicationTaskTag. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/vTaskSetApplicationTag.html

[62] REAL TIME ENGINEERS. xTaskGetApplicationTaskTag. *Freertos.org* [online]. [cit 2014-3-16]. Dostupné z: http://www.freertos.org/xTaskGetApplicationTaskTag.html

[63] SOURCEWARE. The Newlib Homepage. *sourceware.org* [online]. [cit 2014-3-16]. Dostupné z: http://sourceware.org/newlib/

[64] EMBEDDED. Embedding with GNU: Newlib. *Embedded* [online]. [cit 2014-3-16]. Dostupné z: http://www.embedded.com/design/prototyping-and-development/4024867/Embedding-with-GNU-Newlib

[65] EMBEDDED. Embedding GNU: Newlib, Part 2. *Embedded* [online]. [cit 2014-3-16]. Dostupné z: http://www.embedded.com/electronics-blogs/industry-comment/4023922/Embedding-GNU-Newlib-Part-2

[66] KRZYZANOWSKI, Paul. *Process Scheduling* [online]. February 19, 2014 [cit 2014-03-21]. Dostupné z: http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

[67] MAJ, Cezary. *FreeRTOS – API* [online]. [cit 2014-3-21]. Dostupné z: http://fiona.dmcs.pl/~cmaj/ARM/FreeRTOS%20API.pdf

[68] REAL TIME ENGINEERS. FreeRTOS Queues. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/Embedded-RTOS-Queues.html

[69] REAL TIME ENGINEERS. FreeRTOS BinarySemaphores. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/Embedded-RTOS-Binary-Semaphores.html

[70] REAL TIME ENGINEERS. FreeRTOS Counting Semaphores. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/Real-time-embedded-RTOS-Counting-Semaphores.html

[71] REAL TIME ENGINEERS. FreeRTOS Mutexes. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/Real-time-embedded-RTOS-mutexes.html

[72] REAL TIME ENGINEERS. FreeRTOS Recursive Mutexes. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/RTOS-Recursive-Mutexes.html

[73] REAL TIME ENGINEERS. Software Timers. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/RTOS-software-timer.html

[74] REAL TIME ENGINEERS. The timer service/daemon task, and the timer command queue. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/RTOS-software-timer-service-daemon-task.html

[75] REAL TIME ENGINEERS. One-shot timers versus auto-reload timers. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/One-shot-Vs-auto-reload-real-time-software-timers.html

[76] REAL TIME ENGINEERS. Resetting a software timer. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/Resetting-an-RTOS-software-timer.html

[77] REAL TIME ENGINEERS. Memory Management. *Freertos.org* [online]. [cit 2014-3-21]. Dostupné z: http://www.freertos.org/a00111.html

[78] LASZLO, Zoltan. *Memory Allocation in VxWorks 6.0* [online]. Copyright © 2005 Wind River Systems, Inc [cit 2014-3-21]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9869&rep=rep1&type=pdf

[79] REAL TIME ENGINEERS. Memory Protection Unit (MPU) Support. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/FreeRTOS-MPU-memory-protection-unit.html

[80] REAL TIME ENGINEERS. Tasks and Co-routines. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/taskandcr.html

[81] REAL TIME ENGINEERS. Co-routines. *Freertos.org* [online]. [cit 2014-3-28]. Dostupné z: http://www.freertos.org/croutine.html

[82] DI SIRIO, Giovanni. Performance Data. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-4]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:metrics

[83] DI SIRIO, Giovanni. Supported Architectures. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-4]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:architectures

[84] DI SIRIO, Giovanni. ChibiOS/RT Brief Introduction. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-4]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:documents:introduction

[85] MIGLIAVACCA, Martino. *Embedded Systems RTOS: The ChibiOS/RT Project* [online prezentace]. November 18, 2011 [cit 2014-4-4]. Dostupné z: http://home.deib.polimi.it/bellasi/lib/exe/fetch.php?media=teaching:2012:rtos_chibios_handout.pdf

[86] DI SIRIO, Giovanni. ChibiOS background and history. *ChibiOS/RT RTOS* [online forum]. Jun 15, 2012 [cit 2014-4-4]. Dostupné z: http://forum.chibios.org/phpbb/viewtopic.php?f=2&t=481

[87] DI SIRIO, Giovanni. License. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-4]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:license

[88] DI SIRIO, Giovanni. Features Matrix. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-5]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:matrix

[89] DI SIRIO, Giovanni. ChibiOS/RT General Architecture. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-15]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:documents:architecture

[90] DI SIRIO, Giovanni. Threads. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group__threads.html

[91] DI SIRIO, Giovanni. Streams and Files. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group__streams.html

[92] DI SIRIO, Giovanni. Debug. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group__debug.html

[93] DI SIRIO, Giovanni. Registry. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group__registry.html

[94] DI SIRIO, Giovanni. HAL. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group___i_o.html

[95] DI SIRIO, Giovanni. Downloads. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-5]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:download

[96] DI SIRIO, Giovanni. Various. *ChibiOS/RT* [online]. [cit 2014-4-5]. Dostupné z: http://chibios.sourceforge.net/html/group__various.html

[97] DI SIRIO, Giovanni. Configuration: Kernel. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group__config.html

[98] DI SIRIO, Giovanni. Configuration: HAL. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/group___h_a_l___c_o_n_f.html

[99] DI SIRIO, Giovanni. How to create a thread. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-5]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:howtos:createthread

[100] FENG, Lu. ChibiOS Essential: Introdiction to ChibiOS kernel functions, ver 1.0 [online]. [cit 2014-4-5]. Dostupné z: http://www.jacobfeng.com/wp-content/uploads/2013/05/chibios_essential.pdf

[101] DI SIRIO, Giovanni. Kernel Concepts. *ChibiOS/RT* [online]. [cit 2014-4-15]. Dostupné z: http://chibios.sourceforge.net/html/concepts.html

[102] DI SIRIO, Giovanni. Priority Levels. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-5]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:kb:priority

[103] DI SIRIO, Giovanni. Counting Semaphores. *ChibiOS/RT* [online]. [cit 2014-4-7]. Dostupné z: http://chibios.sourceforge.net/html/group__semaphores.html

[104] DI SIRIO, Giovanni. Counting Semaphores, Binary Semaphores and Mutexes explained. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-7]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:semaphores_mutexes

[105] DI SIRIO, Giovanni. Binary Semaphores. *ChibiOS/RT* [online]. [cit 2014-4-7]. Dostupné z: http://chibios.sourceforge.net/html/group__binary__semaphores.html

[106] DI SIRIO, Giovanni. Mutexes. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__mutexes.html

[107] DI SIRIO, Giovanni. Condition Variables. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__condvars.html

[108] ARPACI-DUSSEAU, Remzi H., Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces: 30 Condition Variables* [online]. [cit 2014-4-8]. Dostupné z: http://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf

[109] DI SIRIO, Giovanni. Event Flags. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__events.html

[110] DI SIRIO, Giovanni. Events Explained. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-8]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:kb:events

[111] DI SIRIO, Giovanni. Synchronous Messages. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__messages.html

[112] DI SIRIO, Giovanni. Mailboxes. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__mailboxes.html

[113] DI SIRIO, Giovanni. I/O Queues. *ChibiOS/RT* [online]. [cit 2014-4-8]. Dostupné z: http://chibios.sourceforge.net/html/group__io__queues.html

[114] DI SIRIO, Giovanni. How to manage memory. *ChibiOS/RT free embedded RTOS* [online]. [cit 2014-4-10]. Dostupné z: http://www.chibios.org/dokuwiki/doku.php?id=chibios:howtos:manage_memory

[115] DI SIRIO, Giovanni. Core Memory Manager. *ChibiOS/RT* [online]. [cit 2014-4-11]. Dostupné z: http://chibios.sourceforge.net/html/group__memcore.html

[116] DI SIRIO, Giovanni. Heaps. *ChibiOS/RT* [online]. [cit 2014-4-11]. Dostupné z: http://chibios.sourceforge.net/html/group__heaps.html

[117] DI SIRIO, Giovanni. Memory Pools. *ChibiOS/RT* [online]. [cit 2014-4-11]. Dostupné z: http://chibios.sourceforge.net/html/group__pools.html

[118] BALÁTĚ, Jaroslav. *Automatické řízení*. 1. vydání. Praha: BEN – technická literatura, 2003. ISBN 80-7300-020-2.

[119] PIVOŇKA, Petr. *Číslicová řídicí technika*. Brno: FEKT VUT v Brně, 2003.

[120] HLAVA, Jaroslav. *PID regulátory – jejich vlastnosti, modifikace a číslicová implementace* [online prezentace]. [cit 2014-5-15]. Dostupné z: http://www.fm.tul.cz/esf0247/index.php?download=822

[121] CASPI, Paul, Oded Maler. *From Control Loops to Real-Time Programs* [online]. [cit 2014-5-15]. Dostupné z: http://www-verimag.imag.fr/~maler/Papers/caspimaler.pdf

[122] ZEZULKA, F., J. PÁSEK, M. FINDURA, V. BRAUN, J. PREČAN: *Automatizace procesů*. Brno: FEKT VUT v Brně, 2012.

[123] LIU, C. L., James W. Layland. Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environment. *Journal of the Association for Computing Machinery, Vol 20, No. 1, January 1973*. [cit 2014-5-15].

[124] GCC ARM EMBEDDED MAINTAINERS. GNU Tools for ARM Embedded Processors. *Launchpad* [online]. [cit 2014-5-15]. Dostupné z: https://launchpad.net/gcc-arm-embedded

[125] SOURCEFORGE. WinAVR. *SourceForge.net* [online]. [cit 2014-5-15]. Dostupné z: http://sourceforge.net/projects/winavr/

[126] HERBERT, Jeremy. *Getting Started with the STM32F4 and GCC* [online]. [cit 2014-5-15]. Dostupné z: http://jeremyherbert.net/get/stm32f4_getting_started

[127] LEBEDA, Martin. Cygwin – Unix ve Windows. In: *ROOT.CZ* [online]. 24. 1. 2005 [cit 2014-5-22]. Dostupné z: http://www.root.cz/clanky/cygwin-unix-ve-windows/

[128] VELEBA, Václav. Číslicová řídicí technika: Počítačová cvičení. Brno FEKT VUT v Brně 2005. [cit 2014-5-15].

[129] DI SIRIO, Giovanni. ADC Driver. *ChibiOS/RT* [online]. [cit 2014-5-15]. Dostupné z: http://chibios.sourceforge.net/html/group___a_d_c.html

Seznam zkratek a symbolů

ZKRATKY

A/D	Analogově/digitální
AHB	Advanced High-performance Bus
ALU	Arithmetic an Logic Unit
API	Application Programming Interface
ARM	Advanced RISC Machine
CAN	Controller Area Network
CCM	Core Coupled Memory
CLI	Command Line Interface
CMOS	Complementary Metal Oxide Semiconductor
CMSIS	Cortex Microcontroller Software Interface Standard
CPSR	Current Program Status Register
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
D/A	Digitálně/Analogový
dBSPL	dB Sound Pressure Level
DMA	Direct Memory Access
DSP	Digital Signal Processing
EDF	Earliest Deadline First
FAT	File Allocation Table
FCFS	First Come, First Served
FIFO	First In, First Out
FPU	Floating Point Unit
FSMC	Flexible Static Memory Controller
GNU	GNU's Not Unix
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
I^2C	Inter-Integrated Circuit
I^2S	Inter-IC Sound
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LR	Link Register
LSB	Least Significant Bit/Byte
LQFP	Low profile Quad Flat Package
MAC	Multiply and Accumulate
MEMS	Micro Electro-Mechanical Systems
MMU	Memory Management Unit
MPU	Memory Protection Unit
Msps	Mega Samples Per Second

PC	Program Counter, Personal Computer
PID	Proporcionálně-Integračně-Derivační
POSIX	Portable Operating System Interface
PSD	Proporcionálně-Sumačně-Diferenční
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RMA	Rate-Monotonic Algorithm
RTC	Real-Time Clock
RTOS	Real-Time Operating System
SDIO/MMC	Secure Digital Input/Output / Multi-Media Card
SIMD	Single Instruction, Multiple Data
SMP	Symmetric Multiprocessing
SP	Stack Pointer
SPI	Serial Peripheral Interface
SPSR	Saved Program Status Register
SRAM	Static Random Access Memory
SWD	Serial Wire Debug
TCB	Task Control Block
TCP/IP	Transmission Control Protocol/Internet Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
USB OTG	USB On-The-Go
WCET	Worst Case Execution Time

SYMBOLY

$e(t)$	regulační odchylka ve spojitém čase
$e(kT)$	regulační odchylka v diskrétním čase
K	proporcionální zesílení regulátoru
T	perioda vzorkování, popř. základní perioda systému regulátorů
T_D	derivační časová konstanta
T_I	integrační časová konstanta
$u(t)$	akční zásah ve spojitém čase
$u(kT)$	akční zásah v diskrétním čase
$\nabla u(kT)$	přírůstek akčního zásahu v diskrétním čase
$w(t)$	žádaná hodnota ve spojitém čase
$y(t)$	výstupní veličina ve spojitém čase
c_i	označení i-tého regulátoru, popř. obecně úlohy RTOS
C_i	doba WCET i-tého regulátoru, popř. obecně periodické úlohy RTOS
P	super-perioda systému regulátorů