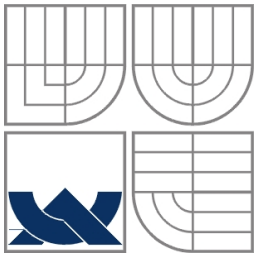


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

USB ZAVADĚČ PRO 8/32 BITOVÉ MIKROKONTROLÉRY

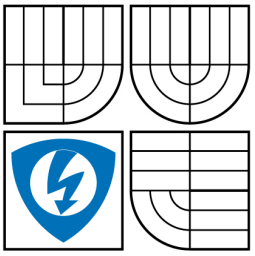
USB BOOTLOADER FOR 8/32 BIT MICROCONTROLLERS

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE — Bc. PAVEL KŘENEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL LAJŠNER
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Diplomová práce

magisterský navazující studijní obor
Mikroelektronika

Student:	Bc. Pavel Křenek	***ID:***	88418
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

USB zavaděč pro 8/32 bitové mikrokontroléry

POKYNY PRO VYPRACOVÁNÍ:

Navrhňete a naprogramujte USB zavaděč pro 8 bitový mikrokontrolér MC9S08JM60 a 32 bitový mikrokontrolér MCF51JM128 firmy Freescale na základě aplikačního manuálu (AN2295). USB komunikace pro oba mikrokontroléry bude probíhat na úrovni virtuálního sériového portu s použitím ovladačů společnosti CMX. Pro učel zápisu do paměti flash použijte existující flashovací funkce pro mikrokontrolér MC9S08JM60. Pro mikrokontrolér MCF51JM128 vytvořte nové flashovací funkce na základě referenčního manuálu. Vytvořte novou verzi protokolu zavaděče pro 32 bitové mikrokontroléry. Celé programové vybavení vytvořte pod systémem CodeWarior v jazyku C, popř. v assembleru. Kompletní dokumentaci včetně všech zdrojových kódů programového vybavení odevzdejte v příloze závěrečné práce.

DOPORUČENÁ LITERATURA:

[1] - LAJŠNER, Pavel. Developer's Serial Bootloader for M68HC08 and HCS08 MCUs., AN2295, 2006
[2] - GALLOP, Martyn., McNAMEE, Joanne. HCS12 Load RAM and Execute Bootloader User Guide, AN2546, 2004

Termín zadání:	9.2.2009	***Termín odevzdání:***	29.5.2009

Vedoucí práce: Ing. Pavel Lajšner, Freescale Semiconductors

prof. Ing. Vladislav Musil, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt:

Předkládaná práce se zabývá problematikou obvodového programování s použitím programového zavaděče. Základním zaměřením práce je vytvoření funkčních zavaděčů pro dva typy mikrokontrolérů firmy Freescale. Vytvoření zavaděčů sestává s implementace stávajícího protokolu pro MCU HCS08 a z návrhu nového protokolu pro MCU ColdFire. Nedílnou součástí je studium a tvorba nových flashovacích funkcí pro 32bitové mikrokontroléry. Tato diplomová práce podává ucelenou strukturu informací o použití a hlavní funkci zavaděčů, zaměřených zejména na konkrétní typy mikrokontrolérů.

Abstract:

This work deals with issues of in circuit programming with using a bootloader. The general aim is to create functional bootloader's for the two different types of Freescale microcontrollers Freescale. The creation of protocols consists of the existing protocol implementation for the HCS08 MCU's and design of the new protocol for the ColdFire MCU's. Next part of this work is studying and creating of the new flashing functions for 32bit microcontrollers. This diploma work provides complete structure of information about using and main function of bootloaders, especially for the specific types of microcontrollers.

Klíčová slova:

Zavaděč, obvodové programování, flashovací funkce, komunikace, mikrokontrolér, flash paměť.

Keywords:

Bootloader, in circuit programming, flash function, communication, microcontroller, flash memory.

Bibliografická citace díla:

KŘENEK, P. *USB zavaděč pro 8/32 bitové mikrokontroléry*. Brno, 2009. 80 s. Vedoucí diplomové práce Ing. Pavel Lajšner. Freescale Polovodiče ČR

Prohlášení autora o původnosti díla:

Prohlašuji, že jsem tuto vysokoškolskou kvalifikační práci vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury. 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 § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 29. 5. 2009

…….

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Pavlu Lajšnerovi za metodické a cíleně orientované vedení při plnění úkolů realizovaných v návaznosti na diplomovou práci. Dále děkuji spolupracující firmě Freescale Polovodiče s.r.o., za poskytnutí prostoru a příslušenství k realizaci diplomové práce.

OBSAH

1 Úvod

V době rozvinutého Internetu a stále výkonnější elektroniky jsme si už zvykli na to, že čím dál větší počet zařízení obsahující mikroprocesory (základní desky počítačů, digitální fotoaparáty, mobilní telefony atd.) nabízí možnost si z Internetu stáhnout nejnovější verzi firmwaru a bez nutnosti navštěvovat servisní středisko si jej do zařízení jednoduše nahrát. Součástí těchto zařízení jsou elektronické obvody a systémy řízené pomocí mikrokontrolérů. Tyto jednoduché mikrokontroléry plní funkci jádra celého systému a mají za úkol celkové řízení a správu systému zpracování zdrojového kódu obsaženého v paměti. Aby mohlo zařízení obsahující jednočipový mikrokontrolér vůbec vykazovat nějakou činnost, musí být vybaveno softwarem. Takovému specializovanému softwaru obvykle říkáme firmware. Je to proto, že ho zpravidla vyvíjí a do procesoru instaluje přímo firma vyrábějící dané zařízení a samotný uživatel s tím nemá nic společného. Dříve tomu tak skutečně bylo a změna firmwaru uloženého zpravidla v paměti EPROM se dala realizovat jen ve specializovaných servisech takových elektronických zařízení.

Programování takových obvodů lze zajistit různými způsoby. Kromě hardwarového programování (BDM, SPI, JTAG) existuje i softwarové programování pomocí programového zavaděče dále už jen „bootloaderu“. Bootloader je program umístěný v horní části programové paměti, která má za úkol přijímat kód a zapisovat jej do dolní části paměti. U softwarového (vnitřního) programování je podnět k přepsání programové paměti nebo pojistek dán provedením speciální instrukce. Kód, který chceme nahrát do programové paměti, musí být odněkud získán - narozdíl od hardwarového programování (kde jsou pevně určeny vodiče a protokol) lze použít libovolný prostředek mikrokontroléru (sériová linka, USB, SPI, TWI nebo vlastní rozhraní). Běžící program tedy může například při startu nebo pravidelně za běhu provádět nahrávání nového programu do paměti.

Cílem této diplomové práce je celkové navržení protokolů a naprogramování softwaru (bootloaderu) pro mikrokontroléry společnosti Freescale. Jedná se o dva různé typy mikrokontrolérů rodiny Flexis, 8bitový a 32bitový. V teoretické části bude podrobně rozebírána funkce bootloaderu z různých hledisek. Mezi nejdůležitější části popsané v následujících kapitolách patří vytvoření flashovacích funkcí. V menší míře je také uveden stručný popis jednotlivých mikrokontrolérů. Dalším milníkem při tvorbě bootloaderu je úprava ovládacích funkcí pro komunikaci přes USB sběrnici.

Tato diplomová práce by měla podat neznalému čtenáři ucelené informace o použití a hlavní funkci bootloaderu, zaměřených zejména na konkrétní typy mikrokontrolérů.

2 Teoretická část

Tato část je zaměřena zejména na vysvětlení základních pojmů týkajících se bootloaderu. Dále podává informace o použitých mikrokontrolérech a vývojových prostředcích. Velká část je tvořena rozborem a ukázkami paměťových prostorů použitých mikrokontrolérů.

Co je to bootloader a proč byl vyvinut? Sériový bootloader byl vyvinut pro nahrání a spuštění vlastní uživatelské aplikace, která je představována binárními daty v souboru. Typ souboru použitý pro tento účel je S-RECORD, což je kódování ASCII pro binární data, více v kapitole 2.3.

Bootloader je umístěn ve vyhrazeném místě v paměťovém prostoru většinou na konci paměti a podporuje nejjednodušší sériový protokol, stejně jako standardní sériové terminály používající k řízení stranu PC (host). Aplikace bootloaderu na straně PC načítá soubor S-Record uživatelské aplikace (vygenerovaný např. programem CodeWarrior), který dekóduje a vytváří si obraz daného souboru. Když je proces dekódování souboru S-Record dokončen, navazuje se spojení s mikroprocesorem, který data přijímá a ukládá do paměti flash.

USB bootloader pro rodinu mikrokontrolérů HCS08 a ColdFire umožňuje provádět tzv. programování v obvodu. Tato vlastnost je obrovskou výhodou pro všechny systémy obsahující mikroprocesory, tím že finální obvod, v němž je nahrán bootloader může být tzv. „za běhu“ naprogramován. Při vylepšení stávajícího software pak není nutné měnit hardwarové zapojení. Pro komunikaci koncového zařízení s PC je využíváno asynchronní komunikace přes sběrnici USB. Zaměření této práce je především na vývoj nového typu protokolu pro rozšiřující se škálu mikrokontrolérů Freescale. Na obr. 1 je znázorněno zapojení vývojové desky s mikrokontrolérem a PC pro vývoj i použití ootloaderu.

Obr. 1: Zapojení vývojové desky s PC pro účely bootloaderu [1]

2.1 Základní požadavky na bootloader

- *Využití malého paměťového prostoru* – bootloader by měl zabírat co nejmenší prostor paměti. Různé druhy bootloaderů využívají přes 1kB paměťového prostoru, což je neakceptovatelné např. při vnitřní paměti flash 3kB. Řešením popisující tento dokument je implementace co nejjednodušších vlastností, při zachování plné funkčnosti. Velikost bootloaderů využívající ke komunikaci USB sběrnici je řádově větší, což je způsobeno větší složitostí obslužného software pro tuto sběrnici. Velikosti bootloaderů se pro tyto účely pohybují v rámci do 6kB – 8kB.

- *Využití co nejmenšího množství pinů mikrokontroléru* – tento typ bootloaderu využívá pro komunikaci s PC sběrnici USB. Pro vlastní přenos dat přes USB je používána vrstva „CDC“ (virtualní sériový port) s klasickým čtyř-vodičovým zapojením. Pro spuštění bootloaderu je nutné použití tlačítka popř. jiného způsobu (časové zpoždění), který synchronizuje mikrokontrolér s PC.

- *Transparentnost s ohledem na uživatelský soubor s19 (viz kap. 2.3)* – výsledný zdrojový kód by měl být transparentní k uživatelskému zdrojovému souboru s19. Uživatel nebude nucen modifikovat zdrojový soubor s19 před použitím bootloaderem. U některých řad mikrokontrolérů je však nezbytný zásah do tohoto zdrojového souboru a to z důvodu přemístění vektorů přerušení (HCS08).

2.2 Základní druhy bootloaderů

Pro dnešní mikrokontroléry existuje celá řada různých bootloaderů, které jsou založeny na různých principech (zápisu dat, komunikaci, systému přerušovacích vektorů atd.). V následující části jsou rozebrány a vysvětleny hlavní druhy bootloaderů používaných zejména pro 8bitové mikrokontroléry.

Mezi základní typ bootloaderu patří dělení podle způsobu dodávání dat do MCU. Ve většině případů tuto funkci plní PC, ale najdou se i bootloadery, které používají k inovaci softwaru další mikrokontrolér. Základem pro každý nový software je výstup z programu na PC, což znamená, že každý nový uživatelský program musí být nahrán určitým způsobem z PC. Většina bootloaderů je naprogramována tak, aby co největší část zpracování zdrojového

kódu byla prováděna v PC, je to z důvodu většího prostoru paměti. PC tak ve většině případů tvoří celkové řízení komunikace a posílání dat, je tzv. master.

Podle zdroje uživatelského kódu rozeznáváme:

Bootloader s masterem PC – komunikaci řídí master - PC, který uskutečňuje celkové ovládání a nahrávání software do cílového MCU. Pro komunikaci mezi PC a MCU je možno využít pouze komunikačních protokolů, které jsou podporovány PC a to jsou např. (USB, RS-232, Ethernet, Paralelní port).

Bootloader s masterem MCU - komunikace řídí master - MCU, který uskutečňuje celkové ovládání a nahrávání software do druhého MCU. Pro tento typ bootloaderu se využívají především typy komunikačních protokolů, jako jsou (CAN, RS-232, USB, SPI, I2C). V případě komunikace dvou MCU po sběrnici USB, musí být master vybavený tzv. USB On-The-GO, které umožňuje komunikaci v režimu HOST.

Další rozdělení bootloaderů je podle způsobu nahrávání samotného zdrojového kódu do MCU. Nahrávání probíhá pomocí programovacího rozhraní (BDM, JTAG, MON08 atd.). Tyto programovací rozhraní nabízejí programátorům celou řadu vlastností pro ladění software tzv. odlaďování.

Podle způsobu nahrávání samotného bootloaderu rozeznáváme:

Bootloader programovaný s uživatelským kódem – tento typ umožňuje použití odlaďovacího rozhraní. Uživatelský zdrojový kód je možné nahrát i za přítomnosti bootloaderu aniž by došlo k jeho smazání, které způsobuje příkaz MCU „mass erase“. Příkaz „mass erase“ vykonává kompletní smazání celé paměti flash a je spouštěn při každém novém programování paměti pomocí programovacího rozhraní.

Bootloader programovaný bez uživatelského kódu – samotný booloader je programován do paměti mikrokontroléru pomocí programovacího rozhraní. Po tomto naprogramování nelze již využívat prostředků programovacího rozhraní, software lze přenášet pouze po externích sběrnicích pomocí tzv. obvodového programování (in circuit programming).

Velkým problémem u většiny bootloaderů jsou vektory přerušení. Různé umístění bootloaderu v paměti se může překrývat s tabulkami vektorů přerušení, což vede většinou k přesměrování vektorů přerušení.

Bootloader s řízením vektorů přerušení – dokáže pracovat s aktuálními uživatelskými vektory přerušení a podporuje přesměrování tabulky vektorů přerušení.

Bootloader bez řízení vektorů přerušení – umí nahrávat software pouze bez použití vektorů přerušení a je vhodný jen pro jednoduché typy aplikací nevyužívající tabulku vektorů přerušení.

Chráněný bootloader – využívá blokovou ochranu paměti flash, která chrání samotný bootloader před přeprogramováním nebo smazáním popř. před zásahem uživatelského kódu.

Nechráněný booloader – Tento bootloader není chráněný před přepsáním. K této situaci dochází u starších mikrokontrolérů, které nemají blokovou ochranu paměti. V některých případech se blokové ochrany záměrně nevyužívá, a to z důvodu složitější vnitřní organizace paměti.

2.3 Formát souborů S-record

Formát souboru S-record byl vyvinut společností Motorola jako kódování ASCII pro binární data. Platforma je známá jako SREC nebo S19 formát. Tento soubor má několik výhod binárního formátu. Kódování ASCII umožňuje editaci v jednoduchém textovém editoru. V každém S-record souboru je obsažena délka pro identifikaci dat, které mohou být znehodnoceny při přenosu.

Soubor S-record byl vytvořen v roce 1970 pro mikrokontroléry Motorola 6800. Struktura souboru S-record je zobrazen v tab. 1.

Tab. 1: Struktura souboru S-record.

TYPE	COUNT	ADDRESS	DATA	CHECKSUM

TYPE – skládá se z jednoho znaku „S“ a hodnoty čísla jednoho digitu 0 – 9 definující typ datového pole.
COUNT – znázorňuje počet bajtů, které jsou následovány ve zbylé části S-recordu.
ADDRESS – čtyř, šesti nebo osmi bajtová adresa ukazující na první bajt dat v paměti.

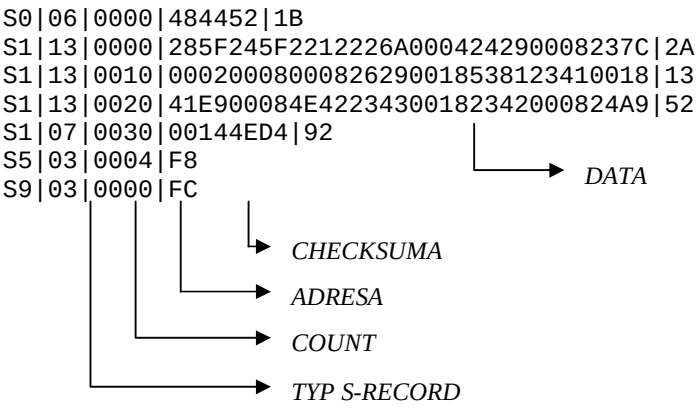
DATA – sekvence 2*n bajtů, pro n bajtů dat.

CHECKSUM – tato suma reprezentuje doplněk nejméně významného bajtu z count, address a datového pole.

Tab. 2: Rozbor jednotlivých souborů S-record

S-record	ASCII kód	Délka adresy	Obsah DAT
S0	0x5330	2 B	Ano
S1	0x5331	2 B	Ano
S2	0x5332	3B	Ano
S3	0x5333	4B	Ano
S5	0x5335	2 B	Ne
S7	0x5337	4 B	Ne
S8	0x5338	3 B	Ne
S9	0x5339	2 B	Ne

Příklad typického formátu souboru S-record:



Soubor S19 podává veškeré potřebné informace o zdrojovém kódu a to ve formě binární struktury S-record. Pomocí této struktury můžeme jednoduše přenést aktuální software z PC do MCU jednoduchým způsobem. Tato struktura je tedy ideální pro úschovu zdrojových dat bootloaderů.

2.4 Mikrokontroléry Freescale řady MC9S08JM

V této podkapitole jsou stručně popsány základní vlastnosti a parametry 8bitových mikrokontrolérů řady HCSS08JM. Jedná se zejména o popis jádra, periférií a hlavní zaměření je kladeno na paměť typu flash. Blokový diagram na obr. 2. zobrazuje celkovou vnitřní organizaci samotného mikrokontroléru.

2.4.1 Základní vlastnosti mikrokontrolérů MC9S08JM60 a MC9S08JM32

Centrální procesorová jednotka (CPU) 8bitového MCU rodiny HCS08

- Frekvence jádra procesoru 48MHz
- Frekvence vnitřní sběrnice 24MHz
- HC08 instrukční sada
- BDM - Background debug mode
- Schopnost vložení "break points"
- Podpora až 32 typů přerušení

Paměťový prostor

- Až 60kB programovatelné flash paměti s blokovou ochranou a nastavením bezpečnosti u mikroprocesoru MCS08JM60 a 32kB u mikroprocesoru typu MCS08JM32
- Až 4kB paměti typu RAM
- 256B paměti USB RAM

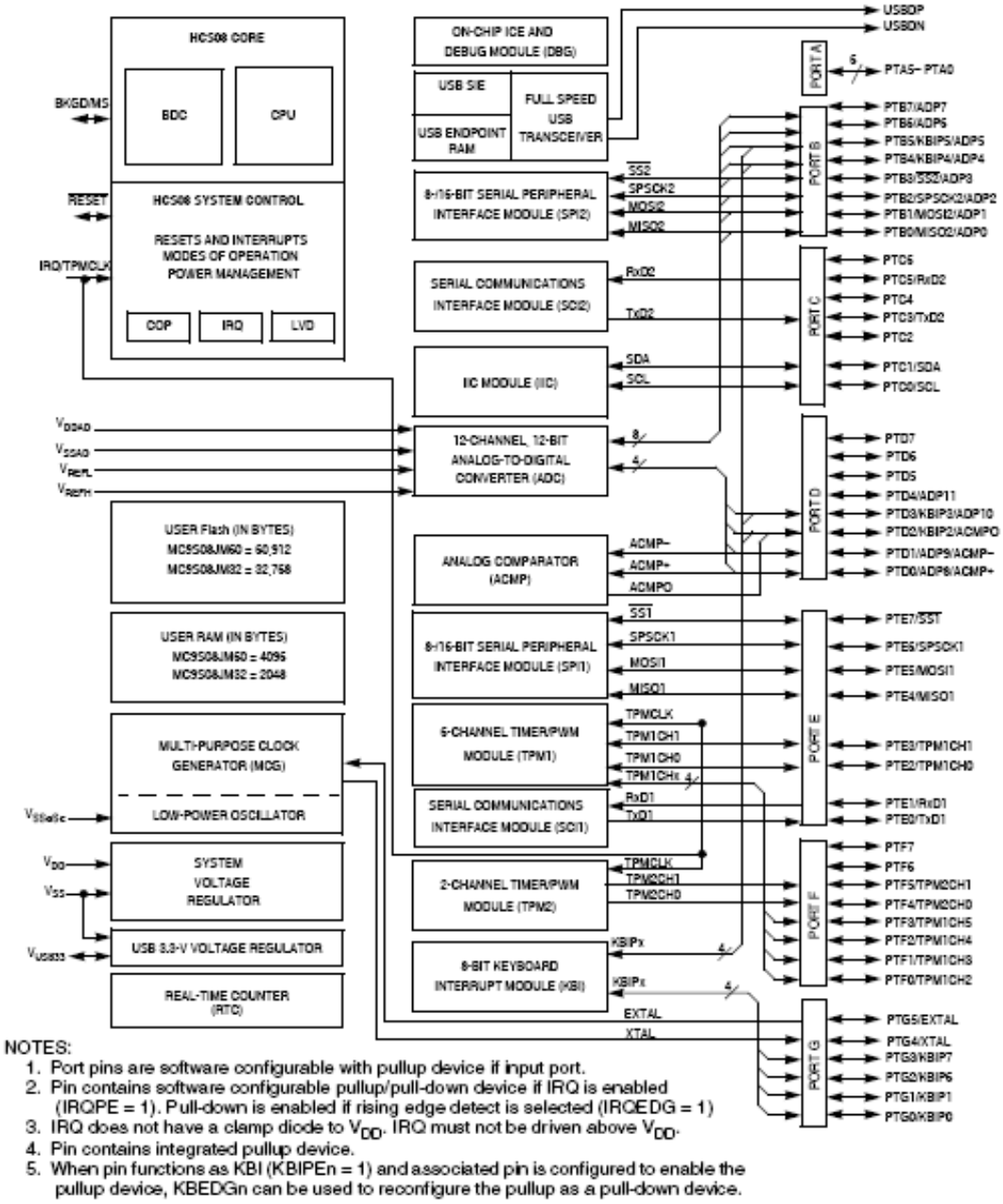
Zdroje hodinového signálu

- Zdroj hodinového signálu (krystal, rezonátor, externí hodiny)
- MCG víceúčelový hodinový generátor
- PLL fázový závěs a FLL frekvenční závěs
- Interní referenční hodiny s přesným doladěním

Moduly periférií

- USB - USB 2.0 full-speed (12 Mbps)
- ADC – 12kanálový, 12bitový AD převodník s vnitřním snímačem teploty
- ACMP - analogový komparátor s nastavením komparace pomocí vnitřní reference
- SCI - dva moduly sériového rozhraní

- SPI - dva 8bitové nebo 16bitové SPI moduly
- IIC - sběrnicový modul pracující na maximální frekvenci do 100 kb/s
- Časovače – 1 x dvoukanálový a 1 x šestikanálový šestnáctibitový časovač

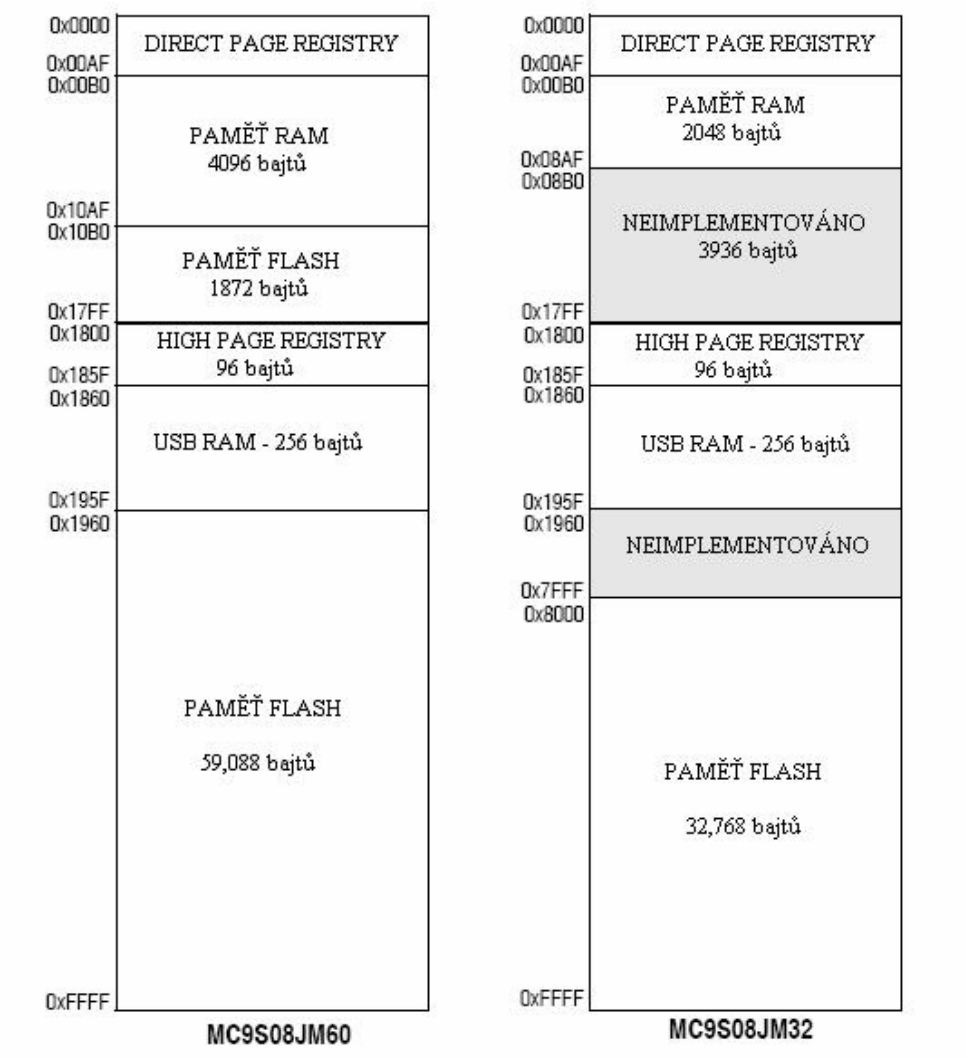


NOTES:
1. Port pins are software configurable with pullup device if input port.
2. Pin contains software configurable pullup/pull-down device if IRQ is enabled (IRQPE = 1). Pull-down is enabled if rising edge detect is selected (IRQEDG = 1)
3. IRQ does not have a clamp diode to V_{DD}. IRQ must not be driven above V_{DD}.
4. Pin contains integrated pullup device.
5. When pin functions as KBI (KBIPEn = 1) and associated pin is configured to enable the pullup device, KBEDGn can be used to reconfigure the pullup as a pull-down device.

Obr. 2: Blokový diagram mikroprocesoru série MC9S08JM[3]

2.4.2 Paměťový prostor mikrokontrolérů MC9S08JM

Paměťový prostor mikrokontroléru MC9S08JM60 je složený z dvou typů pamětí. Jedná se o dynamickou paměť typu RAM a statickou paměť typu flash. V třech blokových segmentech paměťového prostoru se nachází tyto registry: direct-page, high-page a „nonvolatile“. Rozdíly mezi paměťovým prostorem mikroprocesorů MC9S08JM60 a MC9S08JM32 jsou zobrazeny na obr. 3.



Obr. 3: Paměťové prostory mikroprocesorů MC9S08JM60 a MC9S08JM32 [3]

2.4.3 Základní rysy paměti flash řady MC9S08JM

Flash paměť je nevolatelní energeticky nezávislá programovatelná paměť typu RWM. Paměť je vnitřně organizována po blocích a na rozdíl od pamětí typu EEPROM, dovoluje programování po těchto blocích (obsah ostatních bloků zůstává zachován). Paměť se používá jako paměť typu ROM např. pro uložení firmware. Výhodou této paměti je, že ji lze znovu naprogramovat bez vyjmutí ze zařízení s použitím minimálního počtu pomocných obvodů.

Charakteristika paměti flash

- Velikosti pamětí flash
  - MC9S08JM60 — 60kB (119 stránek, každá po 512 bajtech)
  - MC9S08JM32 — 32kB (64 stránek, každá po 512 bajtech)
- Stejné napájecí napětí pro programování i mazání
- Příkazové rozhraní pro rychlejší programování a mazání
- Více než 100 tisíc programovacích/mazacích cyklů pro typickou hodnotu napájecího napětí a teploty
- Proměnné nastavení blokové ochrany paměti
- Automatické vypínání pro nízké frekvence čtení
- Možnost uzamčení flash a ram paměti

Vlastnosti flashovacího rozhraní

Rodina mikrokontrolérů HCS08 je vybavena programovacím rozhraním pro použití „in circuit“ programování. Toto rozhraní zjednodušuje celý systém programování popř. mazání a je založeno na struktuře příkazů zobrazených v tab. 2. Vnitřní obvod programovacího rozhraní obsahuje nábojovou pumpu, která dodává 12V napětí potřebné k programování. Programovaný obvod tak není závislý na připojení externího vyššího napájecího napětí.

Popis registrů programovacího rozhraní

Registr programovacích příkazů (FCMD)

Tento 8bitový registr slouží jako vstupní rozhraní pro rozpoznání aktuálního příkazu. Struktura samotného registru je zobrazena v následující tab. 3. U mikrokontrolérů HCS08 je toto rozhraní složeno z pěti příkazů viz tab. 4.

Tab. 3: Vnitřní struktura registru FCMD

Bit	7	6	5	4	3	2	1	0
Název	FCMD7	FCMD6	FCMD5	FCMD4	FCMD3	FCMD2	FCMD1	FCMD0

FCMD[7-0]: Flashovací příkazové bity - viz tab. 4.

Tab. 4: Příkazy programovacího rozhraní HCS08

PŘÍKAZ	HODNOTA V HEX	STRUČNÝ POPIS
blank check	0x05	Příkaz provedení kontroly paměti flash
byte program	0x20	Příkaz programování po jednotlivých bajtech
burst program	0x25	Příkaz programování většího množství dat najednou
page erase	0x40	Příkaz mazání po stránce o velikosti 512 bajtů
mass erase	0x41	Příkaz kompletního mazání celé paměti flash

Status registr flashovacího rozhraní (FSTAT)

Bity 3,1 a 0 mají vždy hodnotu 0 a zápis do těchto bitů nemá žádný efekt. Zbývajících pět bitů jsou status bity, které mohou být čteny v jakémkoliv časovém okamžiku. Zápis do těchto bitů nemá žádny efekt. Vnitřní struktura tohoto registru je uvedena v tab. 5.

Tab. 5: Vnitřní struktura registru FSTAT

Bit	7	6	5	4	3	2	1	0
Název	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0

FCBEF[7]: (FLASH COMMAND BUFFER EMPTY FLAG) – indikace prázdného bufferu
hodnota 0 – příkazový „buffer“ je plný (není připravený pro další příkaz).
hodnota 1 – nový „burst“ příkaz může být zapsán do příkazového bufferu

FCCF[6]: (FLASH COMMAND COMPLETE FLAG) – dokončení flashovacího příkazu
hodnota 0 – příkaz je právě prováděn
hodnota 1 – všechny příkazy jsou dokončeny

FPVIOL[5]: (PROTECTION VIOLATION FLAG) – porušení blokové ochrany paměti
hodnota 0 – nedošlo k porušení blokové ochrany
hodnota 1 – byl učiněn pokus vymazat nebo naprogramovat chráněné místo

FACCER[4]: (ACCESS ERROR FLAG) – chyba přístupu do paměti flash
hodnota 0 – bez chyby přístupu
hodnota 1 – chyba přístupu byla vyvolána

FBLANK[2]: (FLASH VERIFIED AS ALL BLANK FLAG) – *kontrola* paměti flash
hodnota 0 – indikuje stav, kdy není paměť flash kompletně smazána
hodnota 1 – indikuje stav kompletně smazané flash paměti

Bloková ochrana paměti flash

Základní vlastností blokové ochrany je preventivní ochrana paměťového prostoru před přeprogramováním nebo smazáním. Bloková ochrana je řízena přes registr (FPROT – FLASH PROTECTION). Jestliže je do registru nastavena příslušná hodnota, bloková ochrana začíná od každé 512 bajtové hranice pod poslední adresou flash paměti (0xFFFF).

Během resetu mikrokontroléru se do místa registru FPROT zkopírují hodnoty flash paměti umístěné ve flash oblasti NVPROT. Registr FPROT může být čten v jakémkoliv čase, avšak zapisovat něj lze jen jednou, další zapsání uživatelem při běhu programu nemá žádný efekt. Do NVPROT oblasti může být zapsáno pouze během příkazového ladění, které povoluje přeprogramování, mazání a ochranu flash paměti.

Mechanismus blokové ochrany je zobrazen na obr. 4. Bity FPS jsou používány jako horní bity poslední adresy nechráněné paměti. Adresa je složena z řetězce bitů FPS7:FPS1 s logickou 1. Např. k ochraně posledních 8192bajtů paměti (adresy 0xE000 do adresy 0xFFFF), musí být FPS bity nastaveny na 1101 111, tomu odpovídá poslední adresa

nechráněné paměti 0xDFFF. Kromě nastavení FPS bitů na příslušnou hodnotu, musí být nastaven bit FPDIS na logickou 0, z důvodu povolení blokové ochrany (bit 0 v registru NVPROT). Proto k ochraně paměťového prostoru mezi adresami 0xE000 do 0xFFFF musí být do registru NVPROT vložena hodnota 0xDE. Jeden z nejčastějších způsobů využití blokové ochrany paměti je z důvodu přítomnosti bootloaderu.

Obr. 4: Mechanismus blokové ochrany [3]

Registr ochrany paměti flash (FPROT)

Jediným registrem blokové ochrany je registr FPROT. NVPROT je místo v paměti flash, kde jsou uloženy hodnoty pro registr FPROT viz tab. 6.

Tab. 6: Vnitřní struktura registru FPROT

Bit	7	6	5	4	3	2	1	0
Název	FPS7	FPS6	FPS5	FPS4	FPS3	FPS2	FPS1	FPDIS

FPS [7-1] : p*okud je hodnota bitu FPDIS = 1, pak kombinace 7 bitů v tomto registru definuje poslední adresu nechráněné bloku paměti od nejvyšší adresy v paměti flash. Chráněná oblast nemůže být smazána ani programována*

FPDIS [0] : *hodnota 0 – zapnuta bloková ochrana*
hodnota 1 – vypnuta bloková ochrana

Mechanismus blokové ochrany flash paměti je následující. Kombinace bitů FPS udává poslední nechráněnou adresu v paměťovém prostoru, musí být však nastaven bit FPDIS do hodnoty 0. Příkladem je kombinace nastavení blokové ochrany posledních 8192bajtů v paměti flash (od adresy 0xE000 do 0xFFFF): kombinace FPS bitů pro nastavení registru je 1101111b, výsledná hodnota je 0xDFFF. Samozřejmostí je nastavení bitu FPDIS do 0.

Přesměrování vektorů přerušení

Přesměrování vektorů povoluje uživateli modifikovat informace přerušovacích vektorů bez nechráněného vektorového prostoru a prostoru pro bootloader. Přesměrování vektorů se povoluje pomocí programovatelného bitu FNORED v registru NVOPT umístěným na adrese 0xFFBF v paměti flash. Pro povolení přesměrování vektorů musí být chráněna alespoň nejmenší část paměti blokovou ochranou. Všechny vektory přerušení (v paměti flash na adrese od 0xFFC0 do 0xFFFD) jsou přesměrovány na nastavenou adresu mimo chráněnou oblast paměti, jediný vektor nepodléhající přesměrování je reset, ten zůstává na svém místě.

Registr nastavení paměti flash (FOPT)

Během resetu mikroprocesoru se do místa registru FOPT zkopírují hodnoty paměťového prostoru NVOPT. Tento registr FOPT může být čten v jakémkoliv čase, avšak zapisovat lze jen při restartu MCU, další zapsání uživatelem při běhu programu nemá žádný efekt. Struktura registru je zobrazena v následující tabulce 7.

Tab. 7: Vnitřní struktura registru FOPT

Bit	7	6	5	4	3	2	1	0
Název	KEYEN	FNORED	0	0	0	0	SEC01	SEC00

KEYEN [7] : *0 – backdoor Key Mechanismus povolen*
1 – backdoor Key Mechanismus zakázán

FNORED [6] : *0 – přesměrování vektorů povoleno*
1 – přesměrování vektorů zakázáno

SEC01- SEC00[1-0]: *nastavení bezpečnostní kódu*

2.5 Mikrokontroléry Freescale řady MCF51JM

2.5.1 Základní vlastnosti mikrokontrolérů MCF51JM128 a MCF51JM64

Mikrokontroléry řady MCF51JM jsou součásti rodiny „Flexis“, která byla vytvořena pro jednodušší propojení osmibitových a 32bitových procesorů. Vyšší řada MCF51JM obsahuje stejné periférie jako řada MC9S08JM a obě řady jsou tzv. pinově kompatibilní, což umožňuje jednoduchou možnost migrace z 8bitů na 32bitů.

Mikroprocesory řady MCF51JM128 se skládají z jádra ColdFire Version 1, která je optimalizována pro nižší spotřebu. Tento typ procesoru obsahuje instrukční sadu revize C s redukovaným programovacím modelem. Na obr. 5 je znázorněn blokový diagram zapojení jádra procesorů a periférií.

32bitový Coldfire V1 centrální procesorová jednotka (CPU)

- Frekvence jádra procesoru do 50,33Mhz
- Frekvence sběrnice 25,175Mhz
- ColdFire instrukční sada s redukovaným počtem instrukcí RISC
- BDM - BACKGROUND DEBUG MODE
- Schopnost vložení "break points"
- Podpora až 110 typů přerušení

Uživatelský programovací modul

- 8 datových a 8 adresových 32 bitových registrů
- 32bitový programový čítač
- 8bitový status registr

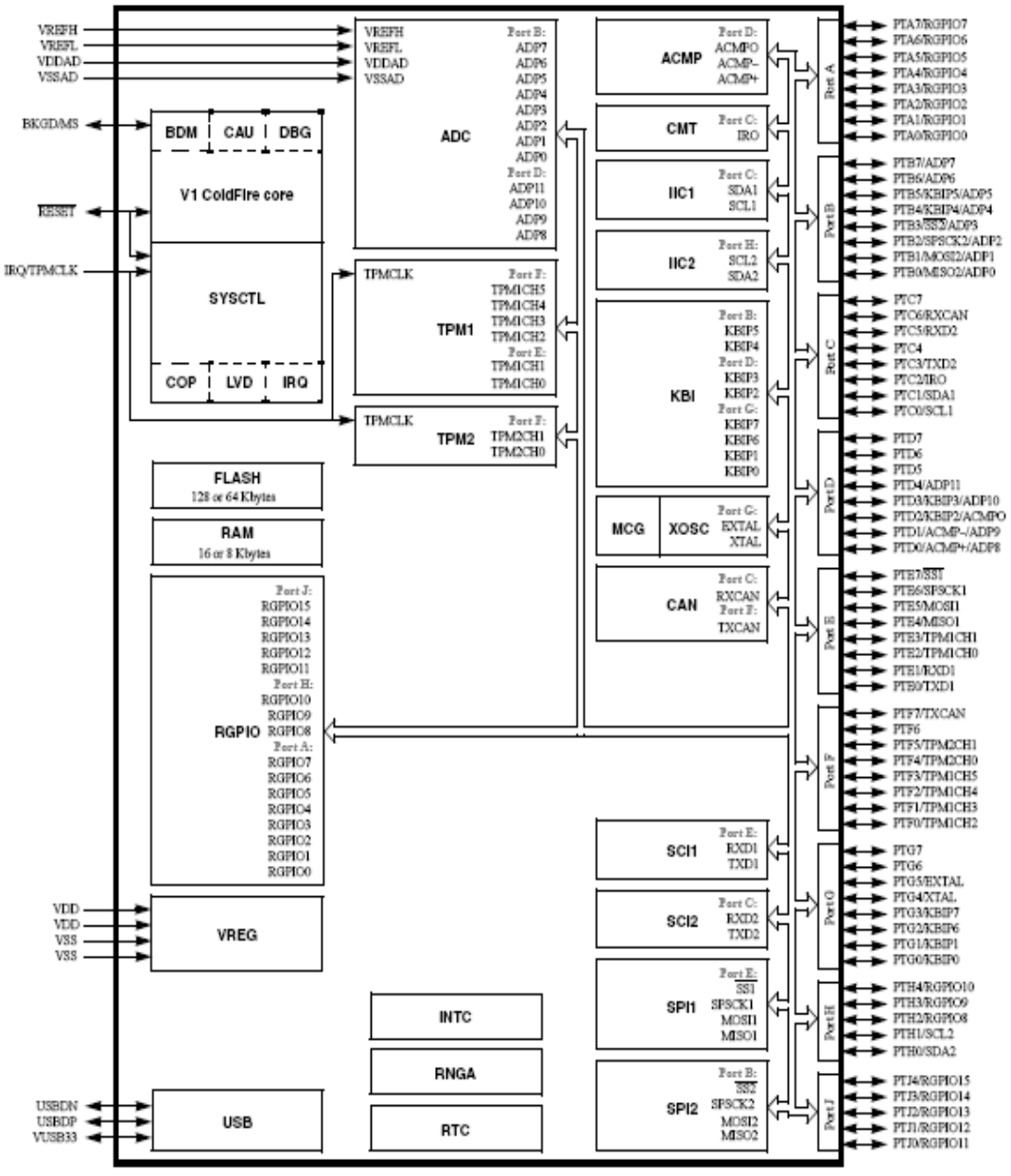
Paměťový prostor

- Až 128kB programovatelné flash paměti s blokovou ochranou a nastavením bezpečnosti u mikroprocesoru MCF51JM128 a 64kB u mikroprocesoru MCF51JM64
- Až 16kB RAM paměti u mikroprocesoru MCF51JM128 a 8kB RAM u mikroprocesoru MCF51JM128

Moduly periférií

- USB - USB 2.0 full-speed (12 Mbps), USB On-The-Go (zařízení i host)
- CAN - podporující standardní CAN komunikační protokol

- 2xSCI - sériová komunikace UART, schopná podporovat RS-232 a LIN protokoly
- 2xSPI - poskytuje 4pinové synchronní sériové rozhraní
- 2xIIC - podporující standardní IIC komunikační protokol
- VREG - napěťový regulátor sloužící ke kontrole a správě napájení celého zařízení
- ADC - 12bitový AD převodník s vnitřním snímačem teploty

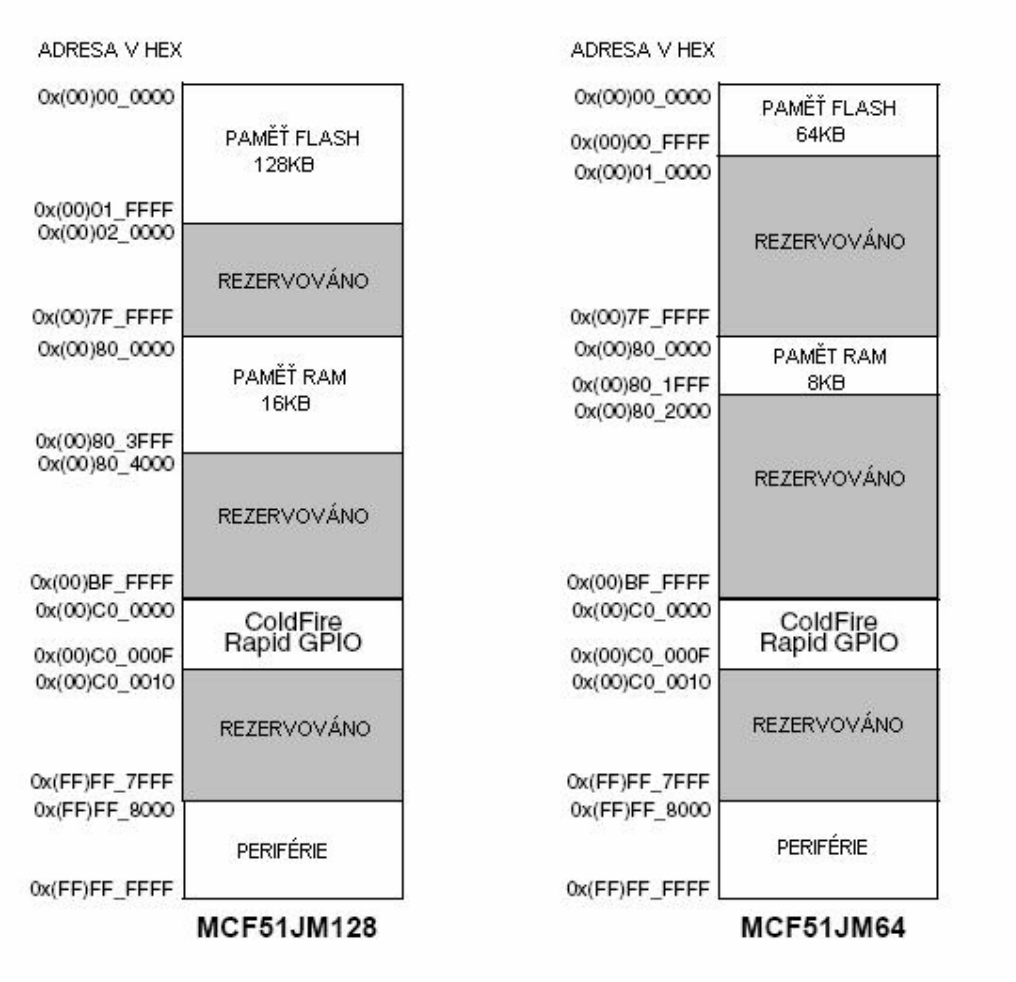


Obr. 5: Blokový diagram mikroprocesoru série MCF51JM [4]

2.5.2 Paměťový prostor mikrokontrolérů řady MCF51JM

Paměťový prostor je u mikroprocesorů řady MCF51JM složený z paměti typu flash pro ukládání statických dat a z paměti typu RAM pro ukládání dynamických dat potřebných pro běh programu. Dále jsou v paměťovém prostoru vymezeny místa pro direct-page registry, high-page registry a pro řízení přerušení viz obr. 6.

- Direct-page registry v rozmezí adres (0x(FF)FF_8000 – 0x(FF)FF_807F)
- High-page registry v rozmezí adres (0x(FF)FF_9800 – 0x(FF)FF_98FF)
- Řízení přerušení (0x(FF)FF_FFC0 – 0x(FF)FF_FFFF)



Obr. 6: Paměťové prostory mikroprocesorů MCF51JM128 a MCF51JM64 [5]

2.5.3 Základní rysy paměti flash řady MCF51JM

Charakteristika flash paměti:

- Velikosti pamětí flash
  - MCF51JM128: 131kB (128 stránek po 1024B)
  - MCF51JM64: 65kB (64 stránek po 1024B)
- Algoritmus automatického mazání a programování
- Rychlé programování a mazání po stránkách
- Možnost rychlejšího programování většího množství dat
- Stejné napájení pro procedury mazaní a programování
- Příkazové rozhraní pro rychlé operace mazaní a programování
- Až 100 tisíc programovacích cyklů při typické teplotě a typickém napájecím napětí
- Pohybliví možnost blokové ochrany paměti
- Systém bezpečnosti pro preventivní neautorizovaný přístup do paměti

Flashovací rozhraní je u MCU ColdFire založeno na stejném principu jako u HCS08, které je rozebráno v předchozí kapitole. Příkazová struktura je zobrazena v tab. 8.

Tab. 8: Příkazy programovacího rozhraní ColdFire

PŘÍKAZ	HODNOTA V HEX	STRUČNÝ POPIS
erase verify	0x05	Příkaz provedení kontroly paměti flash
program	0x20	Příkaz programování po 4 bajtech
burst program	0x25	Příkaz programování většího množství dat najednou
sector erase	0x40	Příkaz mazání po stránce o velikosti 1024 bajtů
mass erase	0x41	Příkaz kompletního mazání cele paměti flash

Bloková ochrana paměti

Základní princip blokové ochrany je stejný jako u HCS08JM viz podkapitola 2.4.3. Bloková ochrana je řízena přes registr FPROT. Tento registr definuje, která oblast paměti má být chráněna před smazáním nebo přeprogramováním. Jestliže v tomto registru nastavena hodnota, bloková ochrana začíná od každé 2048bajtové hranice pod poslední adresou flash paměti (0x1_FFFF).

Během resetu mikroprocesoru se do místa registru FPROT zkopírují hodnoty paměťového prostoru NVPROT. Tento registr může být čten v jakémkoliv čase, avšak zapisovat lze jen jednou, další zapsání uživatelem v běhu programu nemá žádný efekt.

Registr ochrany paměti flash (FPROT)

Tab. 9: Vnitřní struktura registru FPROT

Bit	7	6	5	4	3	2	1	0
Název	FPS							FOPEN

FPS [7-1] : *pokud je hodnota bitu FOPEN = 1, pak kombinace sedmi bitů v tomto registru definuje poslední adresu nechráněné bloku paměti od nejvyšší adresy v paměti flash*

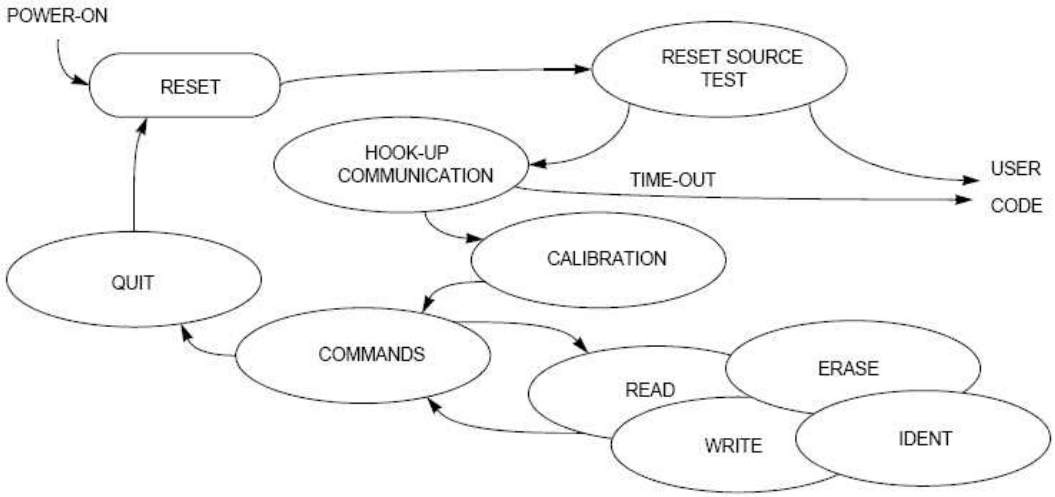
FOPEN [0] : *hodnota 0 – zapnuta bloková ochrana*
hodnota 1 – vypnuta bloková ochrana

Přesměrování vektorů přerušení

Tabulka vektorů přerušení obsahuje 111 vektorů přerušení a začíná na adrese 0x(00)00_0000. Přístup do tabulky vektorů přerušení a přesměrování vektorů přerušení je řízen registrem VBR. Tento registr je 32bitový, pro účely jádra coldfire V1 není implementováno spodních 20B registru. Přesměrování vektorů je možné jen do paměti ram a to od adresy 0x(00)80_0000. Přesměrování může probíhat i při běhu programu a může být neomezeně měněno.

2.6 Vlastní mechanismus bootloaderu

V této části jsou popsány požadavky aplikace bootloaderu a jejich implementace. Implementace „FC“ protokolu pro bootloader musí být jednoduchá a musí zajistit použití co nejmenšího prostoru paměti. Také proto musí být protokol mezi PC (master) a MCU (slave) velmi jednoduchý. Protokol je nazván FC, protože jeden významný znak (potvrzení, nebo ACK) je zde použit ve tvaru 0xFC nebo 11111100b. Na obr. 7 je zjednodušený stavový diagram ukazující oddělené stavy bootloaderu, které jsou popsány v této práci.



Obr. 7: Zjednodušený vývojový diagram pro aplikaci bootloaderu [1]

2.6.1 Inicializace a synchronizace s PC

Existuje několik způsobů pro vstup MCU do bootloaderu. Jedním z řešení může být použití metody logické úrovně přiváděné na vstupní pin MCU.
Příklad může být následující:

- jestliže se objeví na IRQ pinu logická 0 během startu MCU, dojde ke spuštění bootloaderu. Pokud se na pinu logická 0 neobjeví, je spouštěn uživatelský kód. Uživatel má možnost si upravit zdrojový kód bootloaderu, pro bližší specifikaci použitého pinu k spouštění bootloaderu.

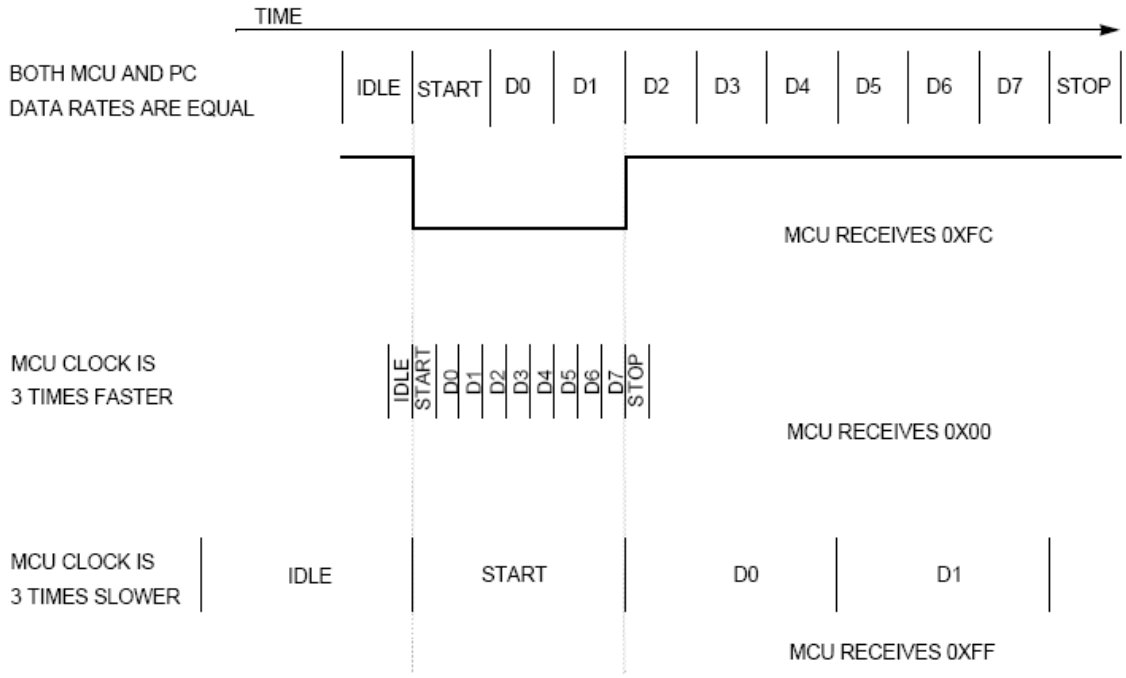
Z požadavků na bootloader vyplývá co nejmenší možný počet obsazených pinů MCU. To znamená, že MCU posílá do PC jeden znak ACK prostřednictvím komunikačního rozhraní a čeká na odpověď. Pokud žádný znak neobdrží ve stanovené době (hook-up time-out), proces pokračuje uživatelským kódem.

2.6.2 Frekvence hodinového signálu

FC protokol umožňuje dva scénáře, a to v závislosti na tom, zda MCU běží na známé a přesné frekvence nebo používá RC (odpor, kondenzátor) hodinový signál nebo vnitřní hodiny MCU (nebo jakékoliv hodinový signál neznámý při kompilaci).

Neznámá komunikační rychlost MCU

Jestliže je frekvence MCU neznámá v době kompilace, MCU nebude kontrolovat potvrzovací znak, pokud nebude odpovídat vzoru $ FC. Vzhledem k toleranci vnitřních hodin MCU může být namísto původního jednoho potvrzení rozesláno více těchto znaků, viz obr. 8.



Obr. 8: Přiřazení různých komunikačních rychlostí [1]

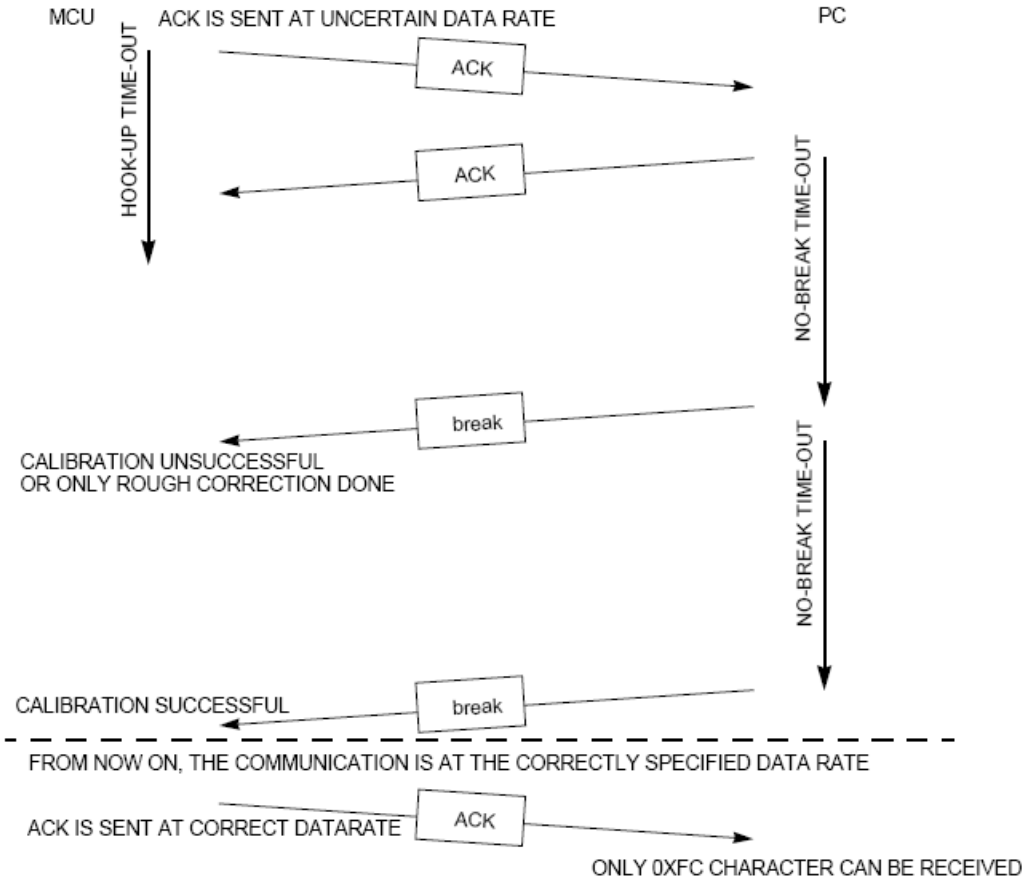
Známá komunikační rychlost MCU

Jestliže je frekvence MCU známa v době kompilace, frekvence MCU bude nakonfigurována tak, aby odpovídala přesně rychlosti komunikace s PC. Všechny znaky jsou přijaty v pořádku a bez zkreslení. MCU posílá $FC do PC, a ten posílá zpět potvrzení do MCU. Po přijmutí potvrzení strana MCU vstupuje do fáze kalibrace.

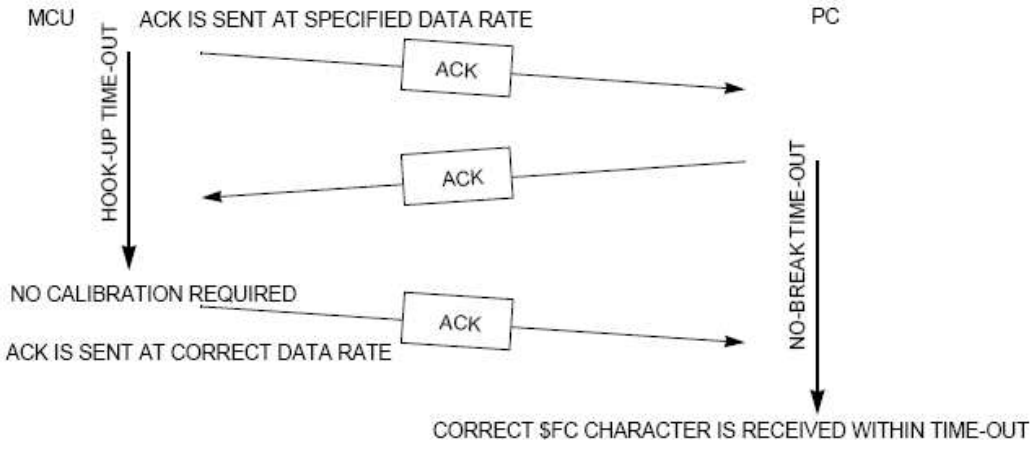
Fáze kalibrace hodinového signálu MCU

Během této fáze se kalibruje vnitřní hodinový signál MCU. Do tohoto okamžiku PC komunikovalo s MCU na rychlosti, která by mohla být od 33% do 300% tolerance. Během této fáze musí být komunikační rychlost MCU upravena tak, aby rychlosti komunikace PC. Po vstupu PC do kalibrační fáze je generováno časové zpoždění. Není-li v tomto časovém intervalu přijat správný znak ($ FC), posílá se tzv. znak „break“ na stejné komunikační rychlosti. Tento znak se skládá z 10 po sobě jdoucí logické nuly. Např. na rychlost přenosu dat 9600 baud, trvá puls z přechodu úrovně vysoká-nízká-vysoká 10 x 104 μs = 1,04 ms. Znak „break“ určuje u MCU, zda je jeho čas příliš rychlý nebo příliš pomalý. Posílání tohoto kalibračního znaku lze opakovat tolikrát, kolikrát je to nutné pro dosažení správné rychlosti MCU. Poté co je MCU kalibrován na správný čas posílá směrem do PC potvrzovací znak ($ FC). Strana PC po přijetí tohoto znaku zastavuje posílání kalibračních znaků, celá situace je zobrazena na obr. 9.

Pokud MCU pracuje na správné komunikační rychlosti (není potřeba kalibrace, a hodinový signál MCU je řízen krystalem). PC může okamžitě odeslat potvrzovací znak ($ FC), kalibrační fáze se přeskakuje, viz obr. 10.



Obr. 9: Začátek komunikace mezi MCU – PC a následná kalibrace [1]



Obr. 10: Začátek komunikace mezi MCU – PC, bez kalibrace [1]

2.6.3 Příkazy pro komunikaci PC a MCU

Po zavedení komunikace mezi mikroprocesorem a PC, mikroprocesor vstupuje do části příkazové smyčky. Mikroprocesor vykonává jednotlivé příkazy k přeprogramování vlastní statické paměti. Komunikace je spojena jako mechanismus master-slave. PC posílá dotazy, mikroprocesor je vykonává a vrací zpátky potvrzení po dokončení každého z příkazů. Potvrzení se provádí zasláním jednoduchého kódu „0xFC“.

Minimální soubor příkazů se skládá z:

- IDENT
- QUIT

Dva další jednoduché příkazy jsou používány výhradně k přeprogramování:

- ERASE
- WRITE

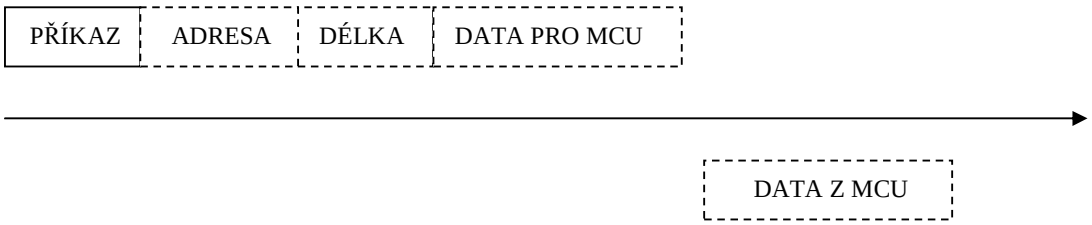
Jestliže uživatel potřebuje provádět kontrolu přeprogramované oblasti je v bootloaderu pro tuto kontrolu implementován příkaz READ. Tento příkaz není nutný k přeprogramování:

- READ

Struktura příkazu pro mikrokontrolér

Na obr. 11 je zobrazena příkazová struktura, přerušovanou čarou jsou vyznačeny části, které nemusí být nutně přítomny při každém příkazu. Data z mikroprocesoru jsou reprezentována jako potvrzení tzv. ACK hexadecimální hodnotou „FC“.

PC posílá příkaz do MCU



MCU posílá odpověď zpět do PC

Obr. 11: Struktura příkazu pro mikrokontrolér

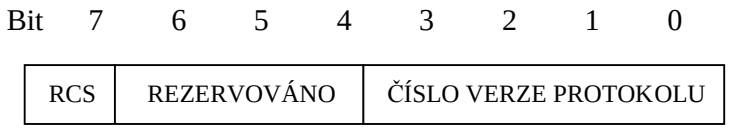
Příkaz „IDENT“

Příkaz ident je definován jako (‘I’, $49)

Tento příkaz je okamžitě vyslán PC stranou po zavedení komunikace, viz obr. 13. Účelem tohoto příkazu je poslat straně PC několik základních vlastností o mikroprocesoru, který má být programován. Všechny více-bajtové informace posílají jako první MSB.

Struktura příkazu „IDENT“

- Verze protokolu



Obr. 12: Zobrazení prvního bajtu příkazu ident

RCS – Flag pro příkaz čtení (podporováno, nepodporováno)

Bit RCS informuje stranu PC, jestli je „read“ příkaz podporován (popř. implementován). Jestliže není, všechna volání příkazu „read“ jsou ignorována a mikroprocesor neposílá zpět PC straně odezvu na tento příkaz. Při tomto stavu software na straně PC varuje uživatele o tom, že nedochází k verifikaci.

BIT [6-4] : *Tyto bity jsou rezervovány pro implementaci nových protokolů pro další MCU a aktuálně jsou nastaveny do 0*

BIT [3-0]: *Tyto bity slouží jako číslo verze protokolu*

- Obsah registru SDID

Registr SDID je šestnáctibitový registr složený z osmibitové části SDIDH a z osmibitové části SDIDL. V tomto registru jsou uloženy hodnoty výrobního čísla mikroprocesoru. Tyto hodnoty jsou unikátní pro každý typ procesoru. Pro implementaci příkazu ident je hodnot v tomto registru využíváno k rozeznání přesného typu MCU.

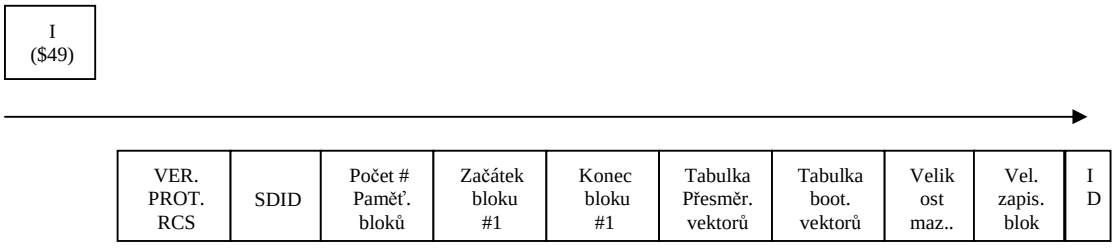
- počet přeprogramovatelných částí paměti flash

Paměťový prostor v MCU bývá často rozdělený na několik paměťový bloků v paměti. Pro tento případ je v protokolu implementován počet přeprogramovatelných částí paměti. Tato podmínka úzce souvisí se začátkem a koncem přeprogramovatelného paměťového prostoru. Při nastavení více paměťových častí nám automaticky generuje další začátek a konec paměťového prostoru.

- Počáteční adresa přeprogramovatelného paměťového prostoru
- Koncová adresa přeprogramovatelného paměťového prostoru
- Adresa tabulky přesměrovaných vektorů přerušení
- Adresa tabulky vektorů přerušení pro bootloader
- Délka bloku mazání v MCU
- Délka bloku zápisu v MCU
- Identifikační řetězec

Tento řetězec obsahuje detailní informace o daném MCU, které nejsou obsaženy v registru SDID, a to je především velikost paměti flash.

PC posílá příkaz do MCU



MCU posílá odpověď zpět do PC

Obr. 13: Grafické zobrazení průběhu příkazu ident

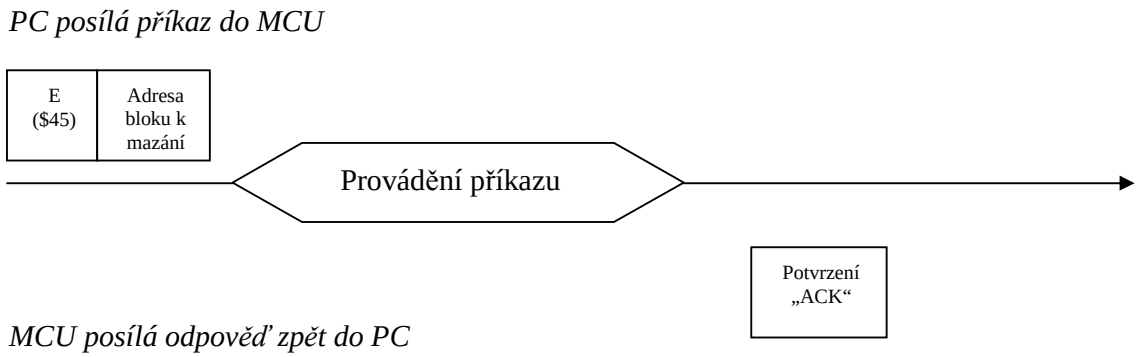
Příkaz „ERASE“

Příkaz „ERASE“ je definován jako (‘E’, $45)

Tento příkaz spolu s ASCII kódem písmena „E“ posílá straně PC počáteční adresu bloku určenému ke smazání viz obr. 14. Počáteční adresa mazacího bloku je dlouhá 2B, s prvním bajtem MSB.

Mikroprocesor smaže paměťový blok odpovídající přijaté adrese. Délka smazané oblasti paměti odpovídá velikosti mazacího bloku.

Poté, co MCU dokončí mazání paměti, pošle potvrzení „ACK“ zpět PC straně. Minimální a maximální časové intervaly průběhu tohoto příkazu nejsou blíže specifikovány.



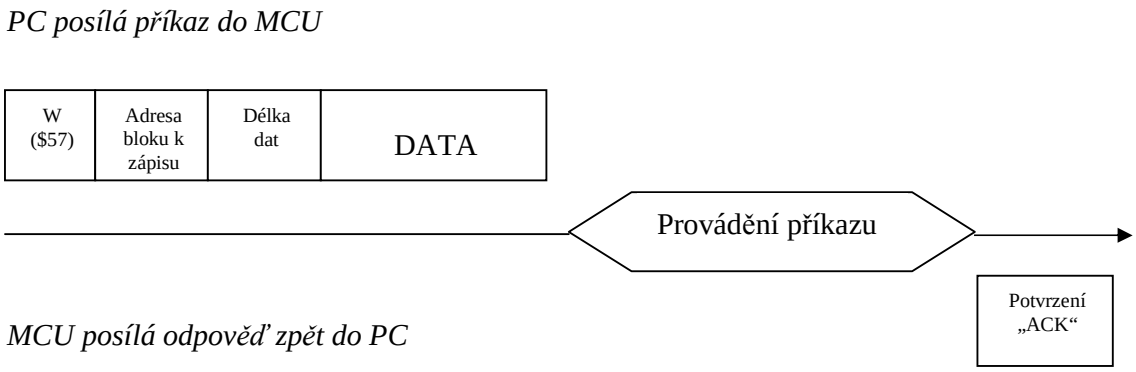
Obr. 14: Grafické zobrazení průběhu příkazu erase

Příkaz „WRITE“

Příkaz „WRITE“ je definován jako (‘W’, $57)

Tento příkaz spolu s ASCII kódem písmena „W“ posílá informace o počáteční adrese, kam má být zapisováno, dále délku dat a samotná data. Počáteční adresa zápisového bloku je dlouhá 2 bajty, s první bajtem MSB. Délka dat je 1 bajt. Posílané data mají omezenou délku podle velikosti zápisového bloku, obsaženou v příkazu „ident“, která je posílána MCU stranou před samotným zápisem, viz obr 15.

Poté, co strana MCU dokončí vykonávání příkazu, pošle zpět do PC potvrzení „ACK“. Minimální a maximální časové intervaly průběhu tohoto příkazu nejsou blíže specifikovány.

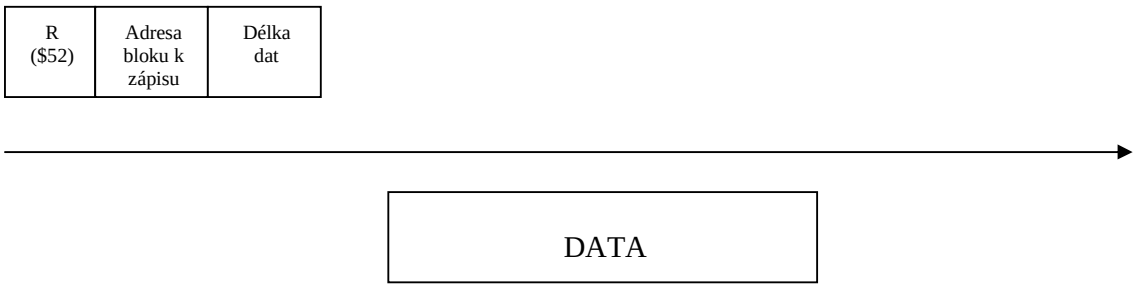


Obr. 15: Grafické zobrazení průběhu příkazu write

Příkaz „READ“

Příkaz „READ“ je definován jako (‘R’, $52)

Tento příkaz spolu s ASCII kódem písmena „R“ posílá informace o počáteční adrese a délce dat, které mají být přečteny. Počáteční adresa zápisového bloku je dlouhá 2B, s první bajtem MSB. Délka dat je 1B. Odezvou MCU na tento požadavek je poslání definovaných dat do PC. Tento příkaz není nezbytný pro funkci protokolu, slouží k verifikaci a je podmínečně kompilovatelný, jeho struktura je zobrazena na obr. 16.



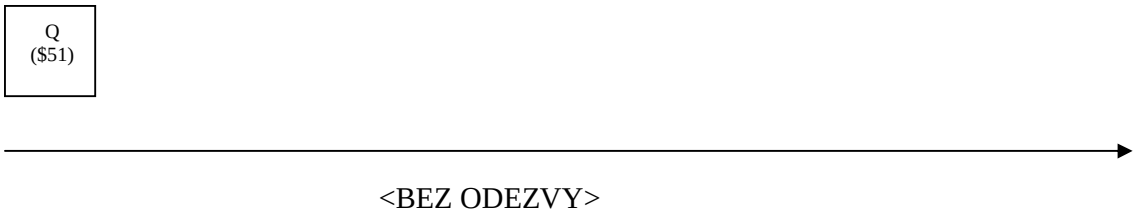
Obr. 16: Grafické zobrazení průběhu příkazu read

Příkaz „QUIT“

Příkaz „QUIT“ je definován jako (‘Q’, $51)

Tento příkaz posílá pouze ASCII kód písmena „Q“. Slouží k ukončení činnosti bootloaderu a přepnutí do uživatelského programu, který byl aktuálně naprogramován. Tento příkaz je bez odezvy ze strany MCU do PC, viz obr. 17.

PC posílá příkaz do MCU



MCU posílá odpověď zpět do PC

Obr. 17: Grafické zobrazení průběhu příkazu quit

2.7 Komunikace po sběrnici USB

USB (Universal Serial Bus) je univerzální sériová sběrnice, umožňující moderní způsob připojení periférií k počítači. Nahrazuje dříve používané způsoby připojení (sériový a paralelní port, PS/2, GamePort apod.) pro běžné druhy periférií. Výhodou je možnost připojování zařízení „plug & play“, bez nutnosti restartování počítače nebo instalování ovladačů. [5]

Základní parametry rozhraní USB:

- Komunikační rychlost od 1,5 Mbit/s do 480Mbit/s.
- Rozhraní obsahuje 5V napájení.
- Lze připojit až 127 zařízení pomocí jednoho typu konektoru.
- USB zajišťuje správné přidělení prostředků (IRQ, DMA).

USB je sběrnice jen s jedním zařízením typu Master, tzn. všechny aktivity vycházejí z PC.
Data se vysílají:

- V krátkých paketech o 8B
- Delších paketech o délce až 256B

Data se přijímají:

- PC může požadovat data od zařízení
- Naopak žádné zařízení nemůže vysílat data samo od sebe

Veškerý přenos dat se uskutečňuje v tzv. rámcích (frame) které trvají přesně 1 milisekundu. Uvnitř jednoho rámce mohou být postupně zpracovávány pakety pro několik zařízení. Přitom se mohou spolu vyskytovat pomalé (low-speed) i rychlé (full-speed) pakety. Obrací-li se PC na více zařízení, zajišťuje jejich rozdělení jako rozdělovač sběrnice (hub). Zabraňuje také, aby signály s plnou rychlostí (full-speed) byly vedeny na pomalá zařízení. Pomalá zařízení pracují s přenosovou rychlosti 1,5 Mb/s. Rychlé přenosy pracuji s rychlosti 12 Mb/s. Časový průběh přenosu informace je předepisován výhradně masterem. Zařízení typu slave se musí synchronizovat s datovým tokem. Používá se k tomu metoda NRZI (Non-Return-To Zero). Nuly v datech vedou ke změně úrovně, jedničky nechávají úroveň beze změny. Kódování a dekódování signálů je čistě hardwarovou záležitostí. Přijímač musí být

schopen získat signál, přijmout a dekódovat data. Speciální prostředky zajišťují, aby nedocházelo ke ztrátě synchronizace.

Obsahuje-li původní datový tok šest po sobě jdoucích jedniček, přidá vysílač automaticky jednu nulu (vkládání bitů), aby se tím vynutila změna úrovně. Přijímač tuto nulu z datového toku opět odstraní. Každý datový paket má za účelem synchronizace speciální zaváděcí bajt (00000001b). Přijímač v důsledku kódování NRZI a vsouvání bitů vidí osm střídajících se bitových stavů, na které se může synchronizovat. Během následujícího přenosu musí synchronizace zůstat zachována. Všechny tyto procesy se odehrávají pouze v odpovídajících hardwarových součástkách. Přijímač a vysílač jsou vždy společně v jedné součástce.

Zařízení USB obsahuje jednotku zvanou SIE Serial Interface Engine, která přebírá vlastní práci. K výměně dat mezi SIE a zbytkem zařízení slouží buffery FIFO. FIFO (First In First Out) to jsou paměti, které mohou postupně přijímat a vydávat data podobně jako posuvné registry. Připojený mikrořadič tedy potřebuje jen přečíst data z FIFO a jiná data do FIFO zapsat. Všechno ostatní vyřídí SIE. Ve většině případů je SIE součástí mikrořadiče USB. Zařízení USB má obecně několik pamětí FIFO, jejichž prostřednictvím je možno přenášet data.[7]

Původním komunikačním rozhraním pro přenos dat starších verzí bootloaderů byla sériová linka RS-232. Tato sběrnice nevyžadovala náročný řídící software, avšak je omezena rychlostí přenosu. Novější sběrnice pro sériový přenos dat, vhodná pro použití ke komunikaci PC s MCU, je sběrnice USB.

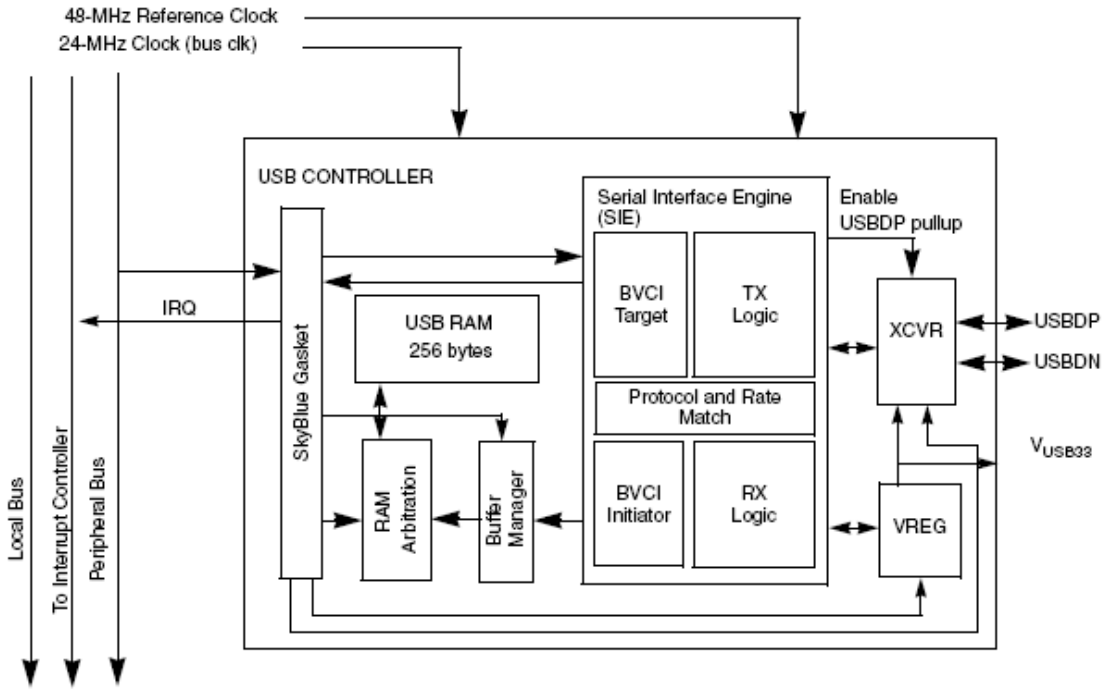
U sběrnice, jako je USB, je nezbytně nutné znát přesný stav procesoru a periferií. Nedbalým naprogramováním bootloaderu může vzniknout řada problémů: povolená přerušení, nastavené směry portů a jiné skutečnosti, které potom způsobují nepředvídané a mnohdy špatně odhalitelné chyby ve vlastním programu.

2.7.1 Popis vlastností USB kontroléru pro HCS08JM a MCF51JM

Flexis rodina mikrokontrolérů disponuje řadičem pro řízení sběrnice USB. Poskytuje jediný čip pro řešením plné rychlosti (12 Mbps) USB zařízení, aplikací a integruje požadovaný vysílač se sériovým rozhraním, 3,3V regulátor, paměť pro koncové body a ostatní ovládací logiku viz obr. 18.

USB řadič zahrnuje následující vlastnosti:

- USB 2.0 plně kompatibilní
  - Přenosová rychlost 12 Mb/s
  - USB datovou kontrolní logiku
    - Identifikace a dekódování paketů (generace)
    - Vytváření a kontrola CRC
    - Kódování / dekódování NRZI
    - Detekce synchronizace
    - Detekce konce paketu
  - 7 USB koncových bodů (endpoint)
    - 6 jednosměrných datových koncových bodů konfigurovatelných jako (interrupt, bulk nebo isochronní)
    - Koncový bod 5 a 6 podporující dvojité bufferování
  - USB RAM
    - Celkem 256 bajtů buffer sdílený se systémem a USB modulem
    - PaměťRAM může být přidělena jako buffer pro USB řadič
  - USB reset volby
    - Reset USB modulu generovaný MCU
    - Reset sběrnice generovaný hostem, který vyvolá přerušení MCU
  - Funkce vysílače
    - Konvertuje USB diferenční napětí na digitální signál logické úrovně
  - Integrovaný pullup rezistor na chipu
  - 3,3V napěťový regulátor



Obr. 18: Blokový diagram USB modulu [3]

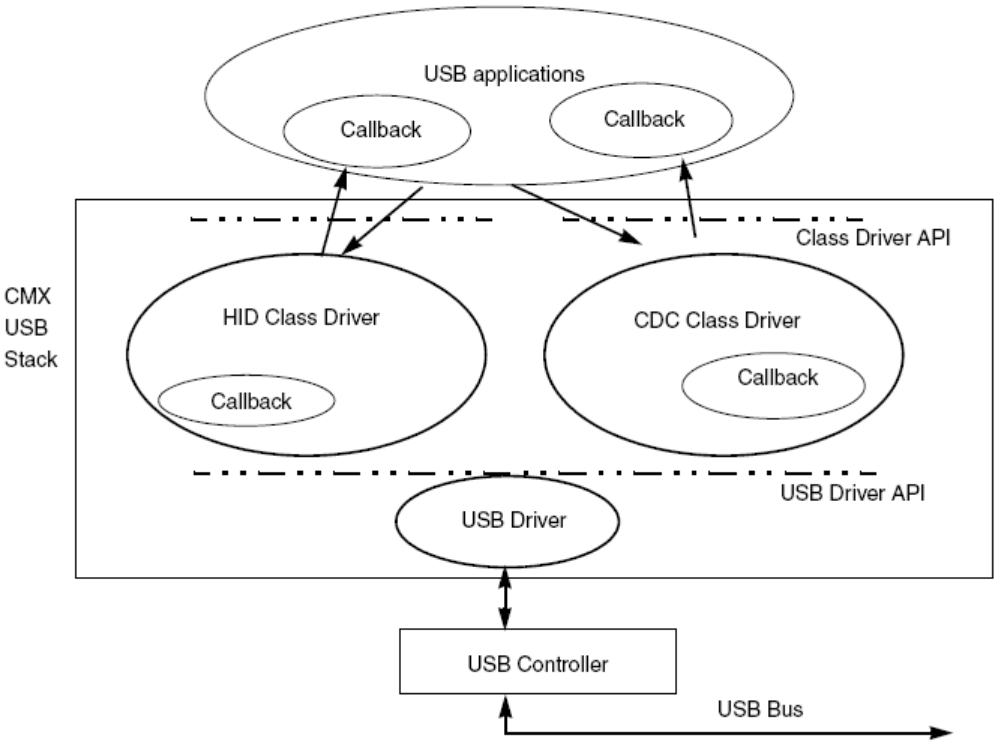
2.7.2 Architektura CMX ovladačů

Pro vlastní komunikaci po USB sběrnici byly vybrány již vytvořené ovladače společnosti CMX, jenž jsou ve verzi Lite volně šiřitelné. Architektura těchto ovladačů je stejná pro oba typy mikrokontrolérů (HCS08JM a MCF51JM) viz obr. 19. Případné úpravy a změny ve zdrojovém kódu jsou diskutovány v kapitole 3.

- **USB ovladač**
  Je základem systému celého USB řadiče, který je řízen protokolem a standardy USB zařízení, stejně jako zprávy vyšší vrstvy třídy ovladače. Při volání událostí se používají systémy návratových funkcí do hlavní aplikace (CALLBACK).

- **Třídy ovladačů**
  Třídy ovladačů HID (HUMAN INTERFACE DRIVER) ovládají protokoly HID, zatímco třídy CDC (CONTROL DEVICE CLASS) řídí komunikační vrstvu protokolu, implementovanou vedlejším řídícím modulem pro sériovou emulaci.

- **USB aplikace**
  Využívají služeb poskytovaných třídami ovladačů k provedení specifických funkcí pro danou aplikaci, založených na USB komunikačním spojení.



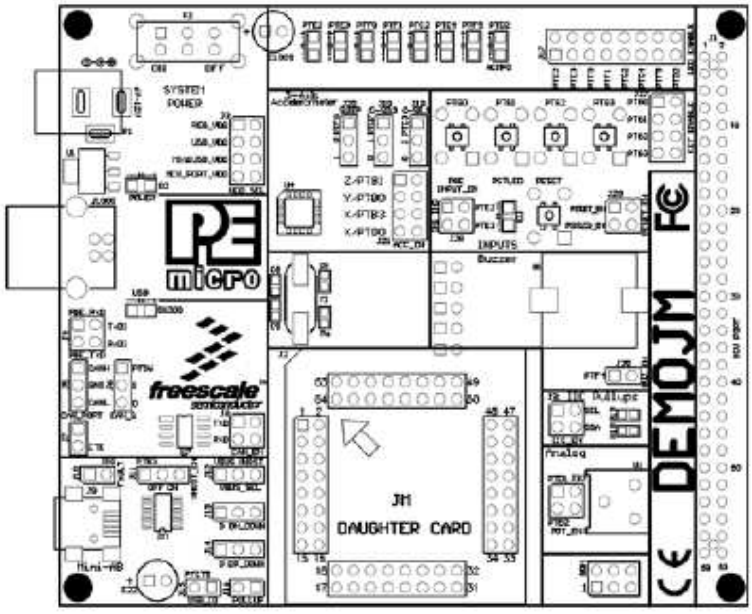
Obr. 19: Architektura USB ovladače (CMX) [6]

2.8 Hardware použitý pro vývoj bootloaderů

Pro účely zkoušení a tvorby bootloaderu byla použita vývojová deska společnosti Freescale DEMOJM. DEMOJM je nízko-rozpočtová systémová podpora mikrokontrolérů MC9S08JM60 a MCF51JM128 s pouzdrem 64LQFP. Skládá se z DEMOJM základní desky a „daughters“ kart DC9S08JM60 a DC51JM128. Na desce je přímo integrován obvod P&E Embedded Multilink, který umožňuje přímé spojení přes USB s PC a slouží zároveň jako programovací, odlaďovací rozhraní. Tato demo deska může být napájena pře USB sběrnici. Na obr. 20 je zobrazena vývojová deska DEMOJM.

Základní parametry vývojové desky DEMOJM:

- Virtuální sériový port
- Obvod P&E’s Embedded Multilink
- SCI signály spojené s Obvod P&E’s Embedded Multilink
- Zařízení USB s konektorem typu Mini-AB
- Modul sběrnice CAN
- XYZ akcelerometr
- 8 uživatelských LED diod



Obr. 20: Pohled shora na vývojovou desku Freescale DEMOJM [10]

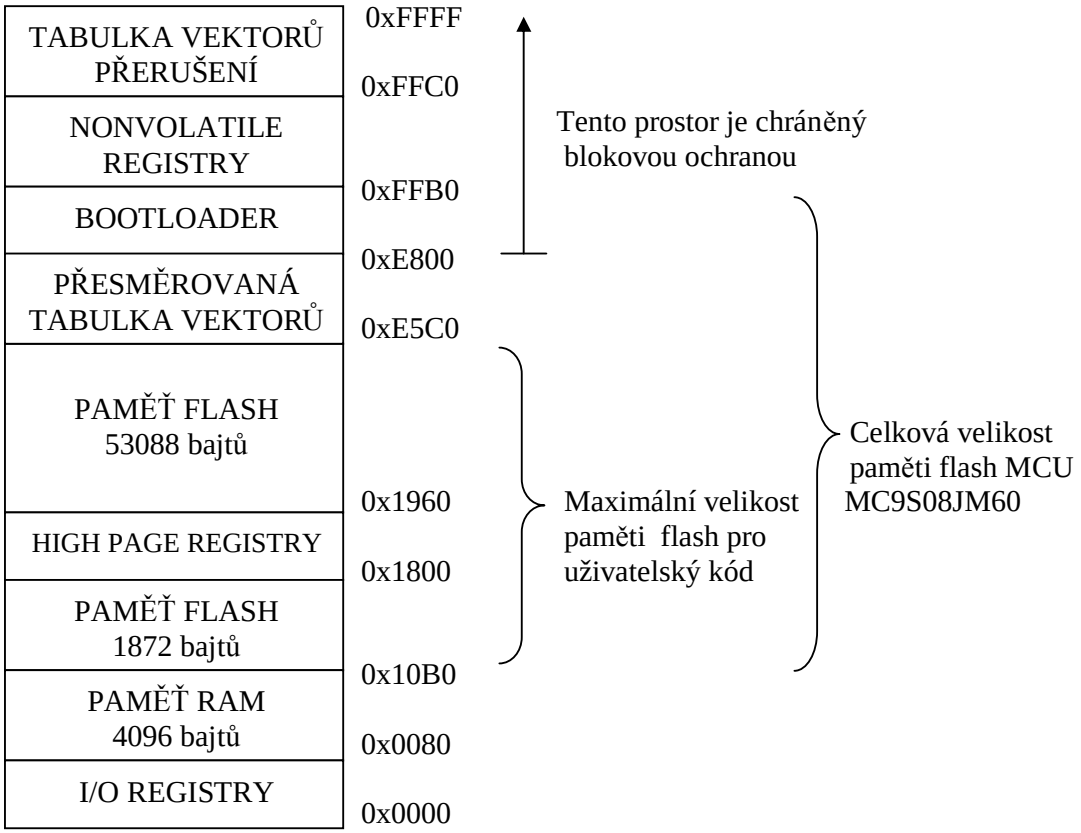
3 Praktická část

Tato část je věnována praktické realizaci a návrhu softwaru pro bootloadery. Důraz je kladen zejména na rozbor činnosti bootloaderů pro dva různé typy mikrokontrolérů a dále pak vysvětlení základních principů flashování a jejich praktickou tvorbu.

3.1 Protokol pro bootloader mikrokontroléru řady MC9S08JM

Tato část popisuje vlastní protokol bootloaderu pro rodinu mikroprocesorů HCS08. Detailní popis tohoto protokolu je prováděn pro mikrokontroléry řady MC9S08JM60. Na obr. 21 je znázorněna mapa paměti pro MCU MC9S08JM60 zahrnující:

- 60kB programovatelné paměti flash
- 4kB paměti typu RAM
- 64B uživatelsky definovaných vektoru přerušení



Obr. 21: Rozložení paměťového prostoru s bootloaderem u MCU MC9S08JM60

Obsazení paměti

Flash paměť mikrokontroléru HCS08JM60 je tvořena dvěmi paměťovými bloky. První část se nachází v rozmezí adres 0x10B0 – 0x1800 s velikostí 1872B. Tento blok paměti není využíván bootloaderem, proto může být plně k dispozici uživateli.

Druhá část paměti se nachází v rozmezí adres 0x1960 - 0xFFFF a její celková velikost je 58kB. V paměťovém bloku v rozmezí adres 0xE800 - 0xFFB0 je umístěn bootloader a těsně pod touto hranicí se nachází přesměrovaná tabulka vektorů přerušení. Zbytek paměťového prostoru tvoří prázdný paměťový blok o velikosti 52kB, který je rovněž plně přístupný uživateli.

Vlastní zdrojový kód bootloaderu je umístěn na nejvyšším místě v paměti flash (na nejvyšším adresovatelném prostoru v paměti). Patřičné umístění umožňuje efektivní využití blokové ochrany paměti flash a neomezuje začátek paměti. Nad tímto místem se nachází primární prostor tabulky vektorů přerušení, jejímž účelem je uložení vektorů přerušení, při neaktivním přesměrování.

Bloková ochrana paměti

Hlavní funkcí blokové ochrany je zamezit uživateli modifikaci popř. smazání vlastního kódu bootloaderu. Po nastavení registru blokové ochrany FPROT a nakopírování jeho obsahu do registru NVPROT, je adresový prostor pod adresou odpovídající nastavení chráněn před smazáním popř. přeprogramováním. Celkové složení paměti v přítomnosti bootloaderu je zobrazeno na obr. 22. Paměťový prostor chráněný blokovou ochranou se nachází v rozmezí adres 0xE800 – 0xFFFF.

Systém přesměrování vektorů

Většina uživatelských programů využívá přerušovacího systému mikroprocesoru. Z tohoto důvodu je nutné zabezpečit chod vektorů v přítomnosti bootloaderu. Menším problémem je situace, kdy samotný bootloader komunikuje pomocí přerušení. Řešením této situace je vytvoření dvou tabulek vektorů přerušení, zvlášť pro samotný bootloader a zvlášť pro uživatelskou aplikaci. Tabulka vektorů přerušení pro bootloader je složena pouze z jednoho vektoru, který je pevně umístěn v paměti flash mikroprocesoru. Přerušovací vektory uživatelské aplikace jsou získávány při každém novém programování z PC.

Tabulka vektorů přerušení pro samotný bootloader je umístěna v chráněné oblasti paměti a je naprogramována pouze jednou spolu s bootloaderem. Naproti tomu tabulka

vektorů přerušení pro uživatelskou aplikaci je umístěna v nechráněném bloku paměti, z důvodu možného přeprogramování. Nejvhodnějším místem pro umístění těchto vektorů je poslední blok nechráněné paměti, kde uživatel nemusí měnit nastavení začátku flash paměti.

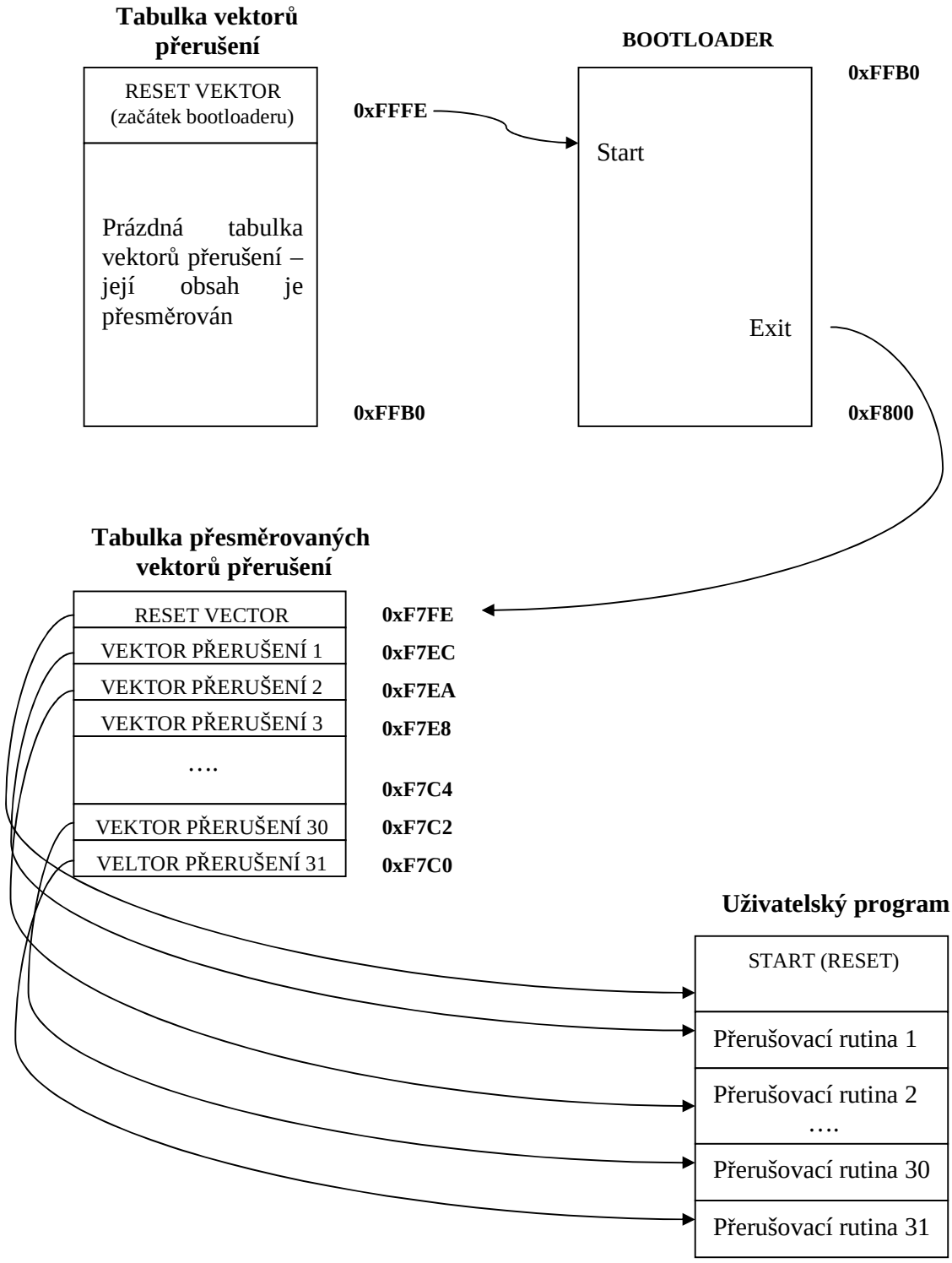
Softwarový reset bootloaderu

Pro ukončení bootloaderu a následném spuštění uživatelského zdrojového kódu, musí být do zdrojového kódu umístěna instrukce pro provádění úmyslně neplatná operace (ILOP-ILLEGAL OPERATION). Pro MCU HCS08JM60 odpovídá této instrukci kód ($8D). Tato ilegální instrukce způsobuje reset mikrokontroléru, což umožňuje transparentní provoz všech ostatních resetů (jako nelegální adresa atd.).

Struktura příkazu „IDENT“

Strana MCU bude posílat tyto následující informace do PC (posílané informace jsou zobrazeny v hexadecimální soustavě):

- Verze protokolu – 0x02
- Obsah registru SDID - 0x016
- Počet přeprogramovatelných částí paměti flash – 0x02
- Počáteční adresa přeprogramovatelného paměťového prostoru #1 - 0x10B0
- Koncová adresa přeprogramovatelného paměťového prostoru #1 - 0x17FF
- Počáteční adresa přeprogramovatelného paměťového prostoru #2 - 0x1960
- Koncová adresa přeprogramovatelného paměťového prostoru #2 - 0xE5C0
- Adresa tabulky přesměrovaných vektorů přerušení - 0xE5C0
- Adresa tabulky vektorů přerušení pro bootloader - 0xFFC0
- Délka bloku mazání v MCU - 512 bajtů
- Délka bloku zápisu v MCU - 64 bajtů
- Identifikační řetězec - "JM60/USB $Version: 1.0.16.0$"



Obr. 22: Přesměrování tabulky vektorů přerušení u bootloaderu pro HCS08JM60

3.2 Master software pro PC

Tato část popisuje software na straně host PC. Všechny zdrojové kódy jsou napsány v jazyku C a jsou kompatibilní s platformou Linux® a Win32®.

Pro účely této diplomové práce byl do tohoto software přidán protokol pro bootloader 32-bitových mikrokontrolérů ColdFire. K naprogramování bylo použito software Microsoft Visual C++ 2005.

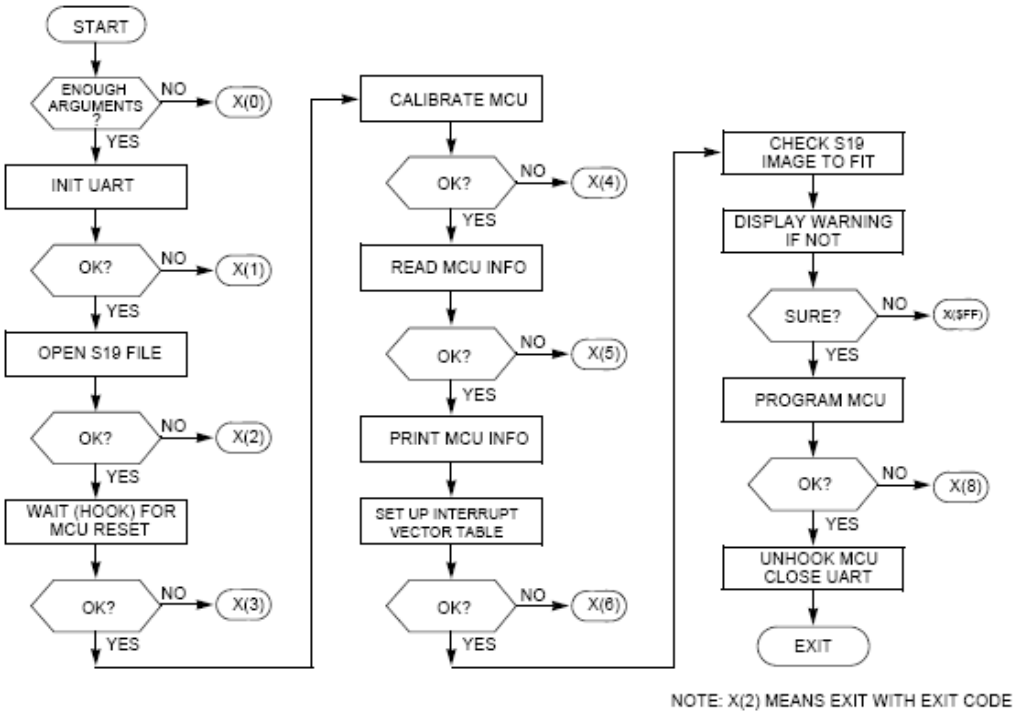
Specifikace bootloaderu udává, aby největší část zdrojového kódu byla prováděna v PC, na místo MCU. Tím dojde k minimálnímu úbytku paměti na straně MCU, kde budou prováděny pouze funkce nezbytné k zápisu a ke komunikaci. Vývojový diagram softwaru pro řídící stranu master je na obr. 23.

Master software je navržen co nejjednodušeji a jeho funkce jsou popsány v následujících krocích:

- Otevření virtuálního sériového portu
- Otevření zdrojového souboru S19
- Čekání na reset od MCU
- Kalibrace MCU
- Čtení informací z MCU
- Manipulace s vektory přerušení
- Kontrola zdrojového souboru S19 ve fyzické paměti
- Mazání a programování MCU
- Vyčištění, konec programu

Otevření virtuálního sériového portu

Master je původně vytvořený software pro komunikaci po klasické sériové lince. Při komunikaci masteru se zařízením USB je chování stejné, jako při komunikaci na sériové lince. Jediným rozdílem je otevírání virtuálního sériového portu, který je vytvořen na straně MCU vlastním USB ovladačem.



Obr. 23: Vývojový diagram software pro master PC [1]

Vytvoření obrazu 8bitového MCU

Master software udržuje binární obraz paměti, který je nezbytný pro operaci se zdrojovým kódem. Informace o aktuálním bajtu v paměti, která má být naprogramována v MCU, je ukládána. nformace jsou získávány ze souboru S19, který je vygenerován pomocí vývojového prostředí pro aktuální nový software.

Tabulka vektorů přerušení

Po přijmutí příkazu „IDENT“ z mikroprocesoru, jsou v dalších krocích prováděny následující operace:

- Zdrojový kód je skenován a zjišťuje se zda-li jsou přítomny případné přerušovací vektory.

- Pokud jsou přítomny vektory přerušení, dochází k jejich přemístění podle specifikace jednotlivých typů procesorů. Původní adresový prostor tabulky vektorů přerušení je označen jako nevyužitý.

Kontrola hranic paměťových bloků

Poslední kontrolou prováděnou před samotným naprogramování MCU je určení, jestli má zdrojový soubor S19 správně deklarované paměťové prostory. Pokud je nalezena některá hodnota mimo rozsah adres mezi začátkem a koncem přeprogramovatelné části paměťového prostoru, je vygenerováno varování.

Mazání a programování MCU

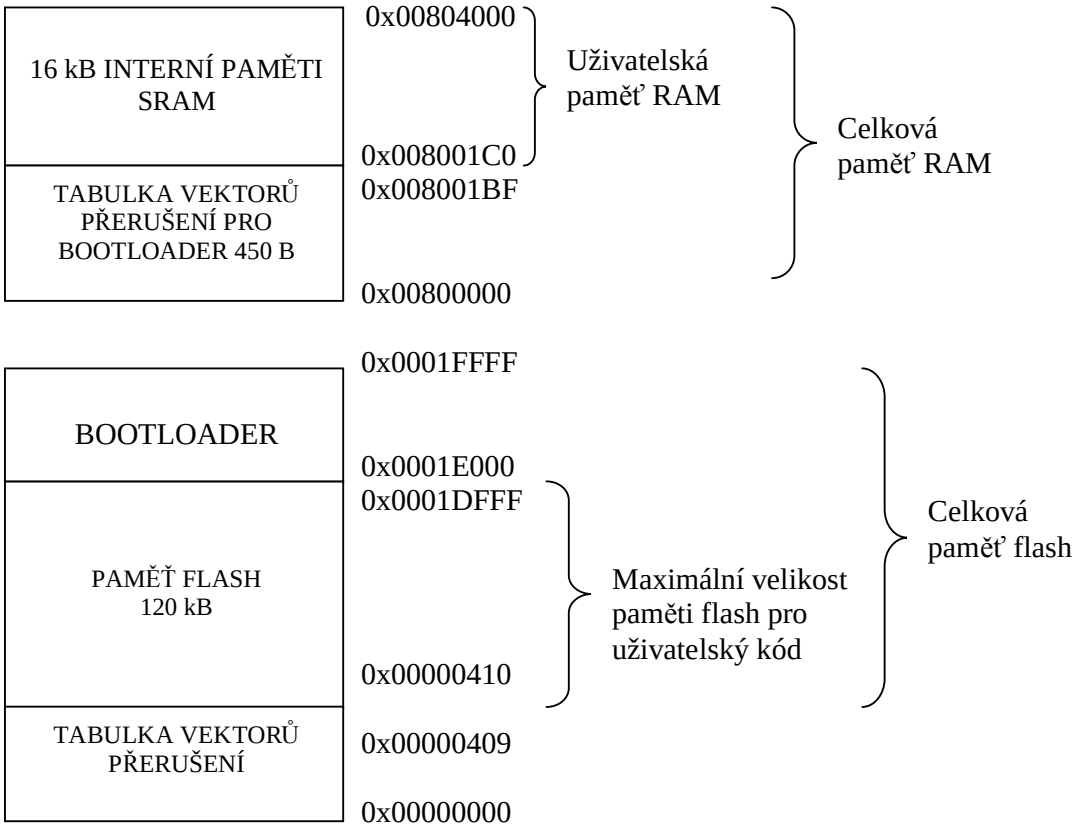
Mazání paměti MCU probíhá po jednotlivých definovaných blocích. PC posílá příkaz „erase“ spolu s počáteční adresou místa, které má být smazáno. Po smazání vymezeného bloku v paměti dochází k naprogramování této části. Velikosti zápisových bloků jsou od 64kB – 128kB, což znamená že master musí posílat několik datových bloku k naprogramování jedné smazané oblasti. Opuštění programu je signalizováno příkazem „quit“.

3.3 Protokol pro bootloader mikrokontroléru řady MCF51JM

Následující část popisuje hlavní strukturu nově vytvořeného protokolu pro 32bitové mikrokontroléry s jádrem Cold Fire V1. Popis protokolu se zaměřuje na typ MCU MCF51JM rodiny Flexis. Pro větší univerzálnost a různé požadavky uživatelů byly vytvořeny dvě verze protokolu pro mikrokontroléry s jádrem ColdFire. Rozdíl mezi verzemi spočívá v principu přesměrování vektorů a použití blokové ochrany paměti. V následující části jsou uvedeny podrobné popisy obou verzí protokolů.

3.3.1 Verze A

Verze protokolu A využívá pro vektory uživatelské aplikace stávajícího prostoru tabulky vektorů přerušení. Díky vhodnému umístění zabírá bootloader méně prostoru v paměti. U tohoto protokolu není použita bloková ochrana paměti flash. Na obr. 24 je znázorněno rozložení paměťového prostoru MCF51JM128 za přítomnosti bootloaderu verze A.



Obr. 24: Rozložení paměti u MCF51JM128 v přítomnosti bootloaderu

Obsazení paměti

Paměť flash je MCF51JM128 tvořena pouze jednou částí o velikosti 128kB. Na začátku této části paměti se nachází tabulka vektorů přerušení, která obsahuje 111 vektorů přerušení. Každý z těchto vektorů má velikost 4B. Samotný bootloader je umístěny v rozmezí adres 0x1E000-0x1FFFF. Zbylá část této oblasti o velikosti 120kB je plně k dispozici uživatelskému kódu. Paměťový blok bootloaderu omezuje paměť flash pouze z horní části a tím umožňuje kompatibilitu zdrojových souborů s19 (nevyžaduje nutnost zásahu uživatele do tohoto souboru).

Paměť ram je pro účely bootloaderu rozdělena do dvou částí. V první části jsou uloženy vektory přerušení pro bootloader a druhá část slouží pro uživatelskou aplikaci. Ukládání vektorů přerušení do paměti ram se využívá z důvodu hardwarového nastavení, který umožňuje přesměrování vektorů pouze do paměti ram.

Systém přesměrování vektorů

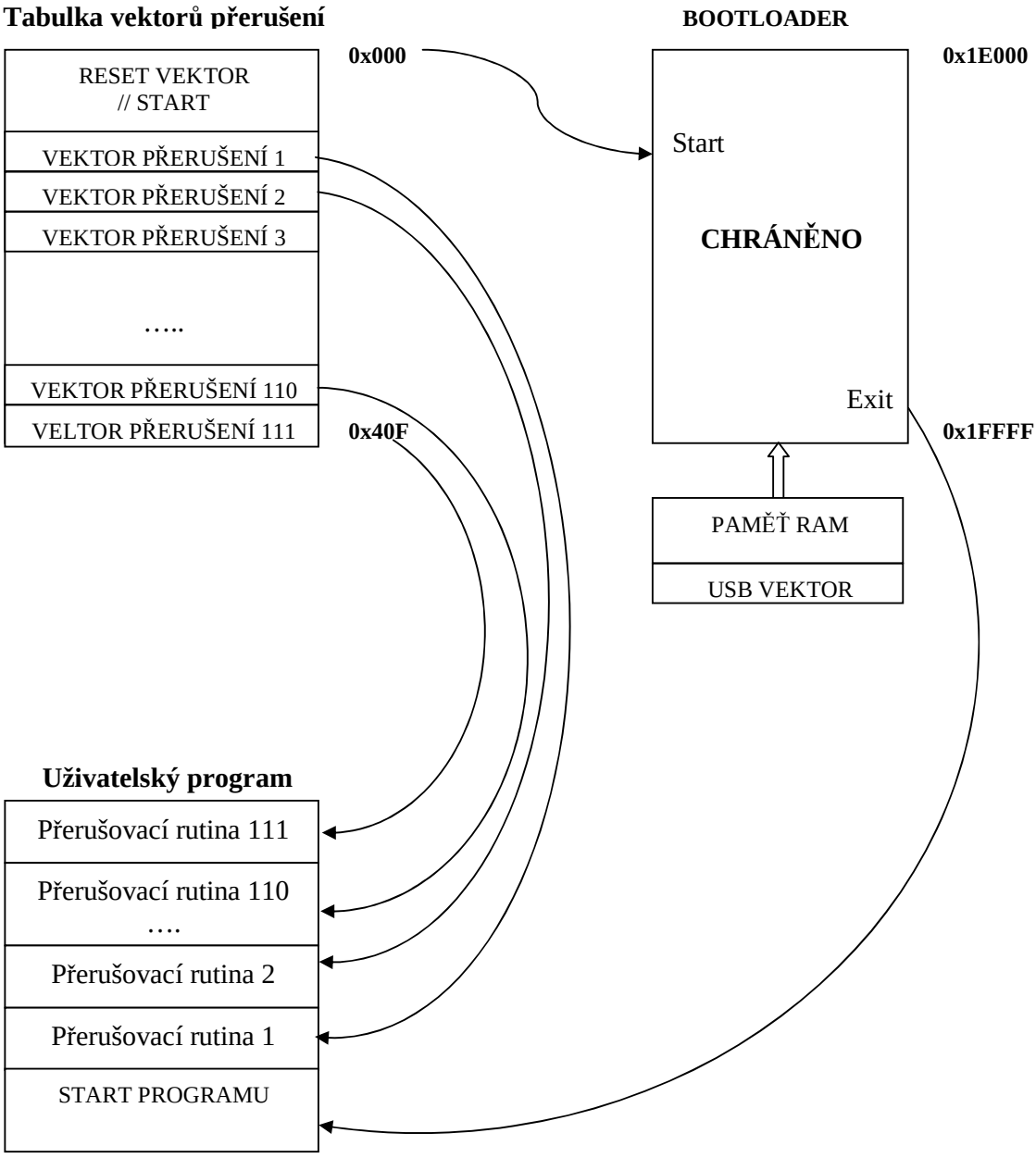
U mikroprocesoru MCF51JM je možné přesměrování vektorů pouze do paměti ram. Tato skutečnost výrazným způsobem omezuje myšlenku předchozího typu protokolu pro rodinu HCS08. Vektory přerušení pro bootloader jsou trvale uloženy v paměti flash. Při každém spuštění bootloaderu dochází k nakopírování vektorů do paměti ram a spuštění přesměrování vektorů. Nové vektory přerušení přenášené z PC se programují na místo originálních vektorů od adresy 0x0 až po 0x1C0. Po ukončení programování vektorů musí být v tabulce uživatelských vektorů přepsán reset vektor, na hodnotu paměťové buňky, na jejíž adrese je umístěn bootloader. Po resetu mikroprocesoru dojde k vypnutí přesměrování a jsou využívány pouze vektory přerušení pro uživatelskou aplikaci. Kompletní situace s přesměrováním vektorů přerušení je zobrazena na obr. 25.

Struktura příkazu „IDENT“

Strana MCU bude posílat tyto následující informace do PC (zobrazení v hexadecimální soustavě):

- Verze protokolu – 0x04
- Obsah registru SDID - 0xC16
- Počet přeprogramovatelných částí paměti flash – 0x01
- Počáteční adresa přeprogramovatelného paměťového prostoru #1 - 0x10B0
- Koncová adresa přeprogramovatelného paměťového prostoru #1 - 0x17FF

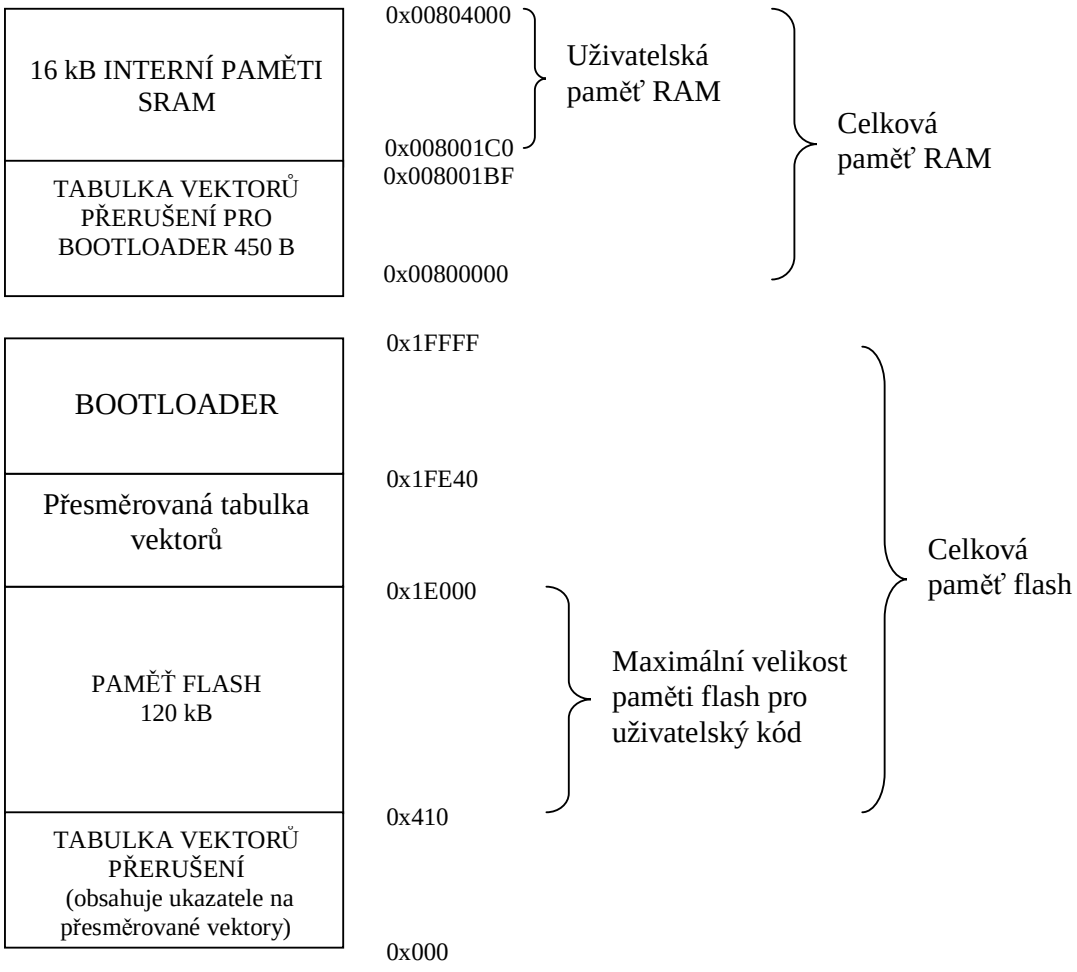
- Adresa tabulky přesměrovaných vektorů přerušení - 0xE5C0
- Adresa tabulky vektorů přerušení pro bootloader - 0xFFC0
- Délka bloku mazání v MCU - 1024 bajtů
- Délka bloku zápisu v MCU - 128 bajtů



Obr. 25: Přesměrování tabulky vektorů přerušení u bootloaderu verze A

3.3.2 Verze B

Verze protokolu B využívá stávající tabulku vektorů přerušení jako prostor, ve kterém jsou uloženy ukazatele na novou tabulku vektorů přerušení. Nevýhodou verze B je větší velikost bootloaderu z důvodu posunutí hranice o velikost tabulky vektorů přerušení. Celkové zobrazení rozložení paměti s bootloaderem je zobrazeno na obr. 26.



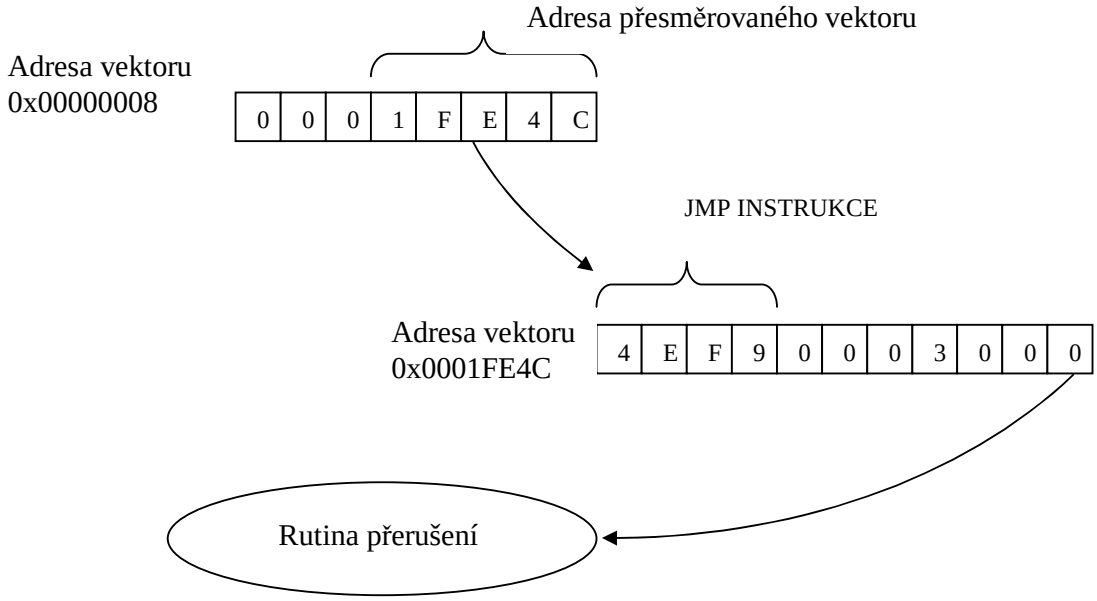
Obr. 26: Rozložení paměti u MCF51JM128 v přítomnosti bootloaderu verze B

Obsazení paměti

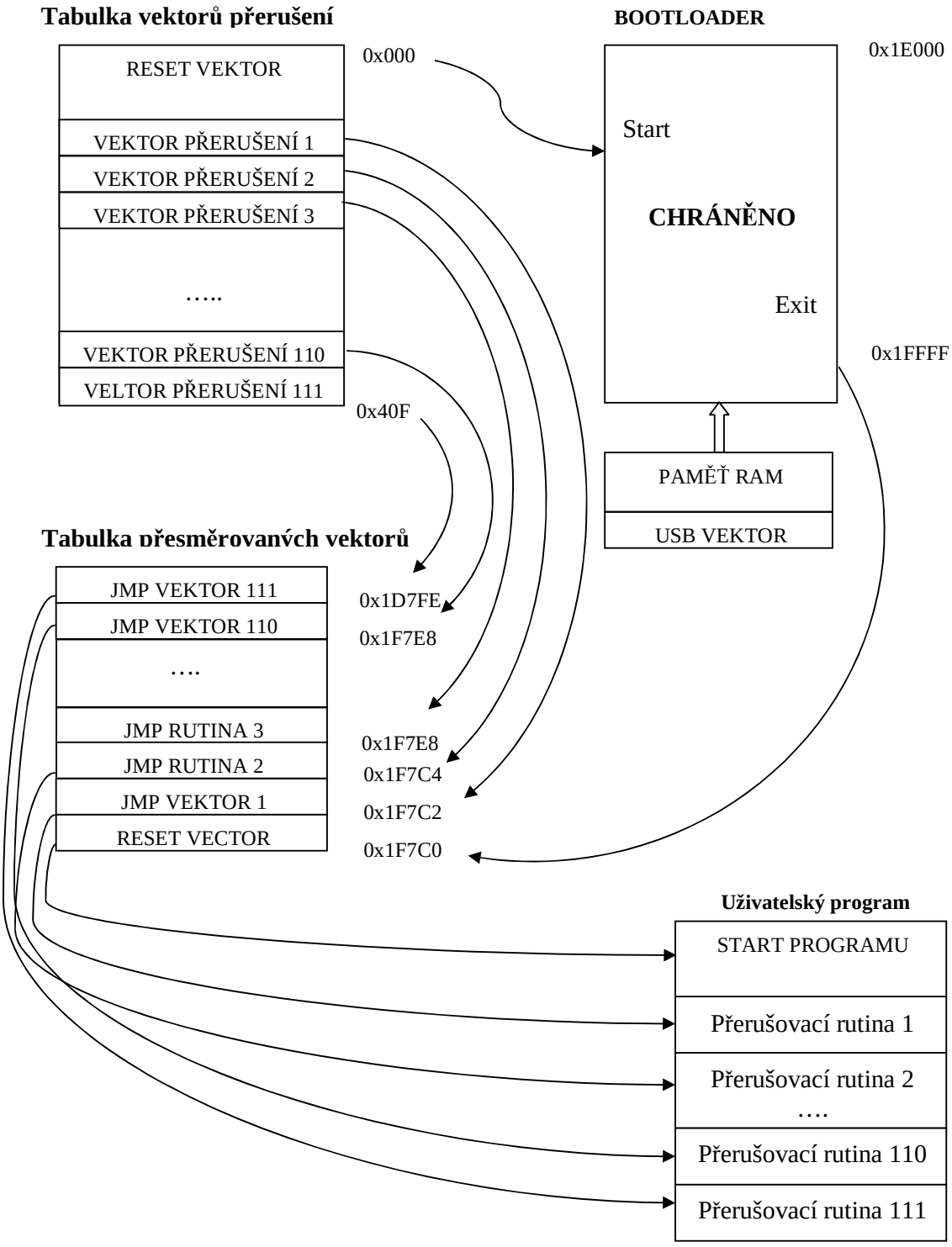
Paměť flash je rozdělena do dvou základních bloků. Prvním blokem je samotný bootloader, který je chráněn blokovou ochranou proti možnému přeprogramování. Druhý blok je nechráněný blokovou ochranou a obsahuje dvě tabulky vektorů přerušení a zbylou část uživatelské paměti flash. Umístění bootloaderu je podmíněno použitím blokové ochrany paměti, která je nastavována od adresy 0x1FFFF směrem dolů po 1kB blocích.

Systém přesměrování vektorů

Z důvodu použití USB komunikace je nutno zavedení vektorů přerušení pro samotný bootloader. Tabulka vektorů přerušení pro bootloader je umístěná pevně v paměti flash a při každém startu bootloaderu je zkopírována do předem definované místo do paměti RAM. Pomocí přesměrování vektorů pak dokáže mikroprocesor využívat tyto vektory pro USB komunikaci mezi MCU a PC. Vektory přerušení pro uživatelskou aplikaci jsou umístěny v posledním nechráněném bloku paměti před bootloaderem, proto aby je bylo možné při každém programování přepsat. Pro správnou funkci vektorů, musí být ve standardní tabulce vektorů přerušení umístěny (od adresy 0x0 hex) ukazatele na nové vektory přerušení. PC strana musí při programování nových uživatelských vektorů zajistit, aby byla před každou adresou vektoru přerušení umístěna instrukce JMP, viz obr. 27. Tato instrukce umožňuje provádět nepodmíněný skok a to přímo na podprogram přerušení. Principiální zobrazení celého systému je na obr. 28.



Obr. 27: Struktura přesměrování jednoho vektorů přerušení



Obr. 28: Přesměrování tabulky vektorů přerušení u bootloaderu verze B

3.4 Vytvoření a popis flashovacích funkcí

Tato kapitola je zaměřena na rozbor tvorby flashovacích funkcí pro jednotlivé typy bootloaderů. Pro botloader mikrokontroléru HCS08JM byly použity již existující flashovací funkce s názvem DoOnStack, které byly patřičným způsobem upraveny.

V další části je rozbor nově vytvořených flashovacích funkcí pro mikrokontroléry řady MCF51JM, které byly vytvořeny v assembleru. Rozbor dané problematiky je provázán ukázkami zdrojového kódu, který slouží pro detailnější pohled na principy flashování.

3.4.1 Flashovací funkce pro MCU HCS08JM „doOnStack“

V doslovném překladu anglického slova „doOnStack“ vyplývá jedna důležitá zásada programování flash pamětí. Každý zápis do paměti popř. mazání vyžadují tzv. řídící software, který bude dohlížet na správnost celého systému programování. U mikrokontrolérů se k ukládání většiny softwaru používá paměti flash. Otázkou zůstává, kam se ukládá software potřebný k zápisu do této paměti? Odpovědí je, že programování pamětí flash se musí uskutečňovat z jiného paměťového prostoru, než z aktuálně programovaného. K tomuto účelu se využívá paměti RAM a blíže pak přímo zásobníku, který slouží pro krátkodobé ukládání dat.

Intervaly mazání a programování paměti

Pro správnou funkci flashovacího rozhraní je nutné správné nastavení vnitřních časových intervalů, které se mění v závislosti na interní frekvenci mikrokontroléru.

Před začátkem každého programovacího nebo mazacího cyklu musí být nastaven registr FCDIV. Do tohoto registru musí být zapsána číselná hodnota odpovídající frekvenci sběrnice mikroprocesoru a zároveň výsledná hodnota frekvence programovacího nebo mazacího cyklu, která by měla být v rozsahu frekvencí od 150kHz do 200kHz. Do registru FCDIV může být zapsáno pouze jednou, po restartu procesoru, většinou v části programové inicializace, viz (1). V případě, že je nastaven bit FACCER - chyba přístupu v registru FSTAT, nemůže být do registru FCDIV zapsána. Uživatel musí zajistit, aby bit FACCERR nebyl nastaven před zápisem do FCDIV. Půl periody výsledné frekvence procesoru je použita jako časový interval k mazání a programování. Tato čtyřbajtová hodnota časového impulsu je použita v příkazu procesoru k dokončení mazání nebo programovaní.

```
FCDIV= 0x4F;   /* FCLK=(PRDCLK)/1+FCDIV[5:0]=3000kHz/(1k+15k)=187,5 kHz
```
(1)

Příkaz mazání a programování paměti

Mazání je u paměti flash možné jen prostřednictvím jednotlivých bloků v paměti. Velikost jednoho takového bloku je u HCS08JM 512B. V případě programování je nejmenší možná velikost paměťové buňky 1B. V následující části je popsána struktura funkce vlastního vytváření a principu flashování. Celý princip flashování je zobrazen na vývojovém diagramu viz obr. 29.

Před vyvoláním každého příkazu musí být nastaven registr FCDIV a smazány všechny chybové bity.

Postup vykonání příkazu je následující:

1. Zapsání datové hodnoty na adresu v poli paměti flash. Informace o adrese a datech musí být předány flashovacímu rozhraní. Tento zápis je požadován jako první krok každé příkazové sekvence. Pro mazací příkazy není hodnota dat důležitá. Adresa musí být v rozsahu 512B stránky, která má být smazána. Před každým zapsáním do paměti flash musí být provedeno její smazání, a to buď celkové pomocí příkazu „mass erase“, nebo po 512B stránkách příkazem „page erase“.

2. Zapsání požadovaného příkazu do registru FCMD.

3. Nastavení bitu FCBEF do 1 v registru FSTAT , následné spuštění příkazu. Před každým dalším spuštěním příkazu musí být FCBEF smazán až poté může být nastaven a vykonávat další možný příkaz.

Po vykonání příkazu musí proběhnout kontrola bitů porušení ochrany FPVIOL a chyby přístupu FACCER. Pokud je nějaký z těchto dvou bitů nastaven celý cyklus se ukončí. Při správném provedení příkazu následuje čekání na znamení od bitu FCCF (bit úspěšného provedení příkazu), pokud má tento bit v daný okamžik hodnotu 1, dochází k ukončení cyklu. Ukázka zdrojového kódu mazaní a programovaní je znázorněna v následujícím textu.

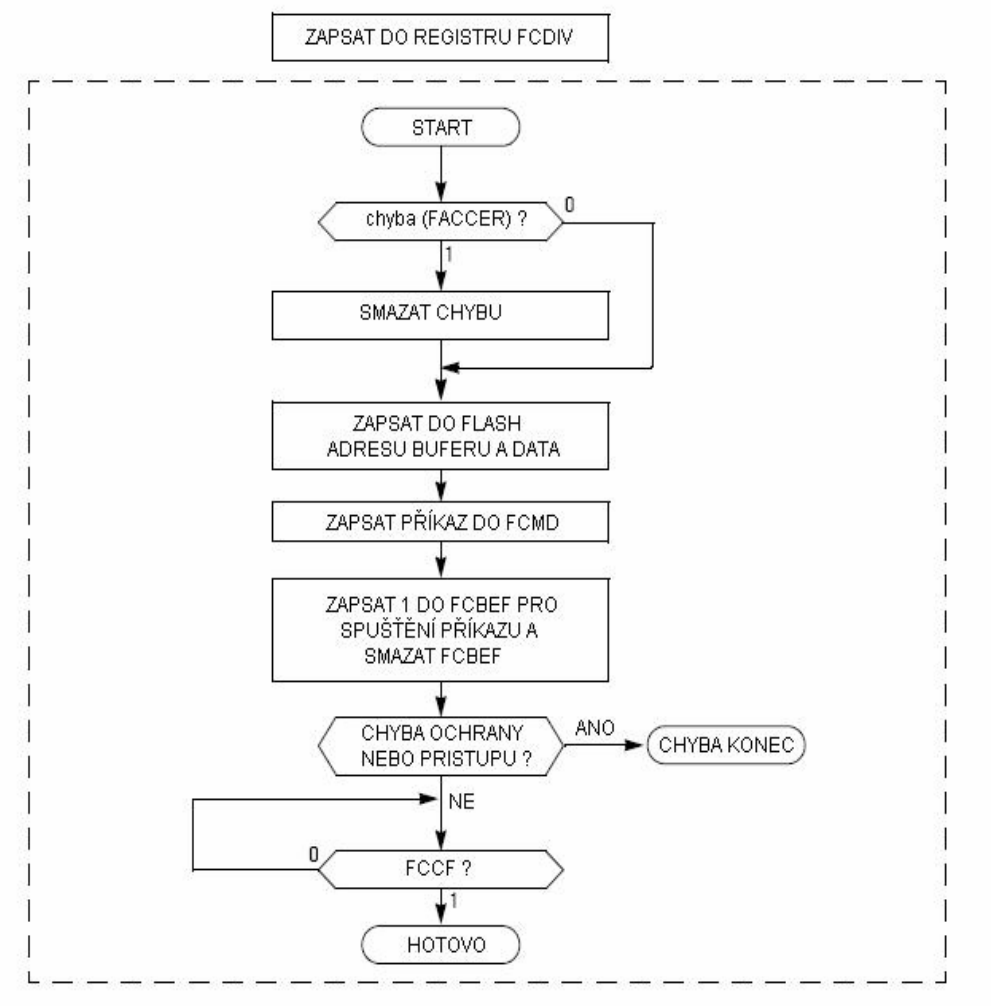
Vlastní zdrojový kód se skládá z jednotlivých funkcí pro mazání a programování. Tyto jednotlivé funkce jsou označeny návěštími se správnými názvy a jsou ukončeny instrukcí rts,která plní funkci návratu do hlavní smyčky programu.

Ukázka zdrojového kódů flashovacích funkcí

V následující ukázce zdrojového kódu je zapsán postup flashovací funkce zobrazený na obr. 29. Programovací jazyk použitý pro vytvoření flashovací funkce je assembler.

```
FlashErase1&FlashProg1:

        psha
        lda #(mFPVIOL + mFACCERR)       - maskování bitů
        sta FSTAT                       - ukládání registru FSTAT
        lda # mPageErase                - načtení požadovaného bitu
        bsr DoOnStack                   - skok na návěští
        ais #1
        rts                             - návrat do hlavního programu
```



Obr. 29: Vývojový diagram pro mazání a programování

Další části zdrojového kódu je funkce, která slouží ke zkopírování aktuálně používané funkce na zásobník. Jednotlivé funkce jsou kopírovány na zásobník, před každým jejich použitím, tím je zároveň šetřeno místo v paměti RAM. U mikroprocesorů s větší pamětí ram je možné nakopírovat všechny funkce do paměti najednou a pak se jen odkazovat na tyto funkce pomocí nepodmíněných skoků. Před tím, než je vlastní flashovací funkce spuštěna musí dojít k uložení návratových adres, na které se program odkáže po skončení flashovací funkce.

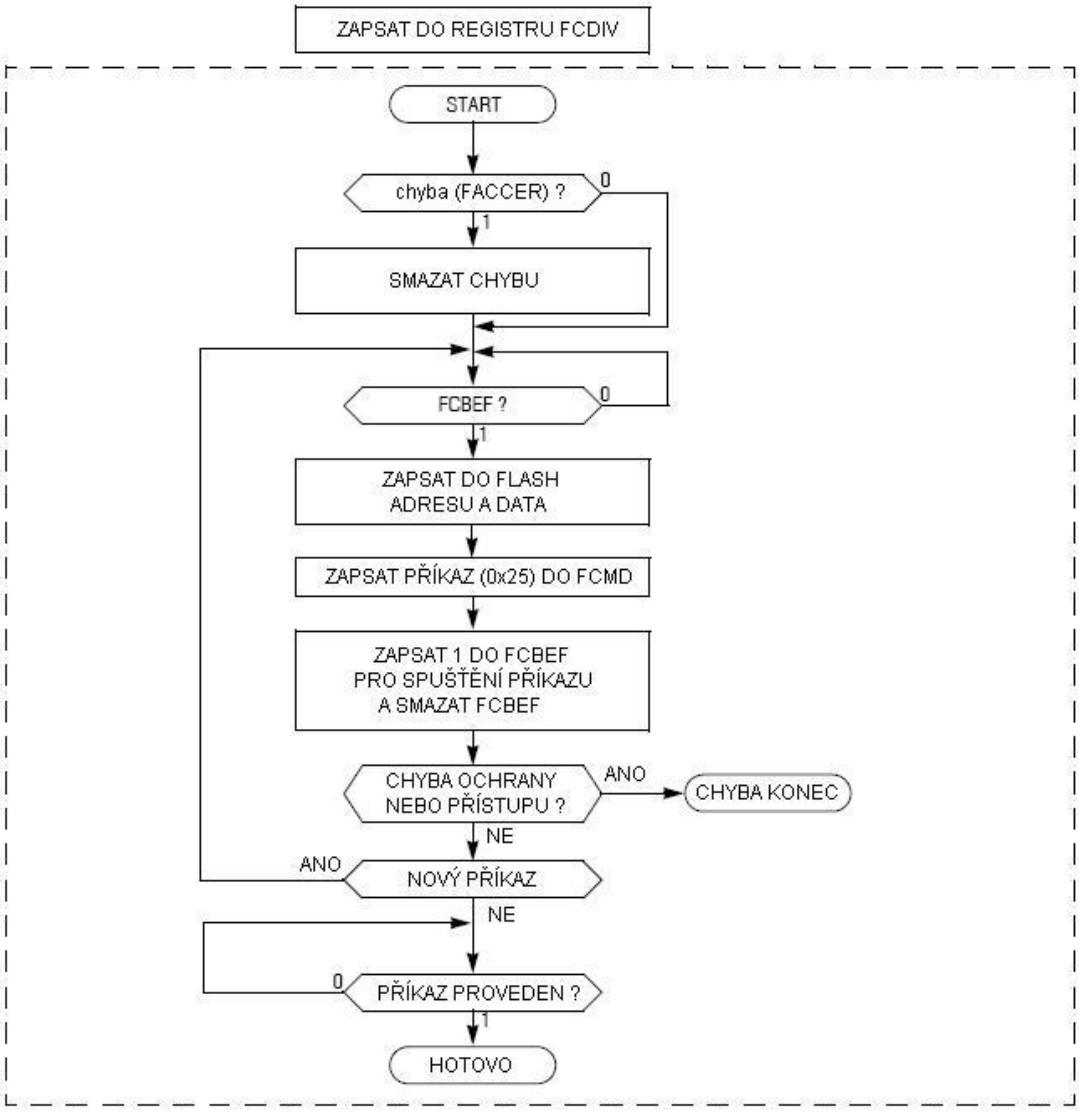
Programování většího množství dat pomocí příkazu „burst“

Tento příkaz se používá pro programování datových sekvencí v kratším časovém intervalu, než by tomu bylo při použití klasického programovacího příkazu. Příčinou tohoto zrychlení je programování většího množství dat najednou ve formě pole, při této operaci nemusí být přerušováno vysoké napětí potřebné k programování. Obvykle, když je použit klasický příkaz mazání nebo programování se musí začít generovat vysoké napájecí napětí potřebné k flashování. K tomuto účelu slouží tzv. nábojové pumpy. Po dokončení jednoduchých flashovacího příkazů se nábojová pumpa vypíná ihned. Při použití programovacího příkazu „burst“ zůstavá nábojová pumpa zapnutá po celou dobu flashování, a to po dobu, kdy jsou splňovány následující podmínky:

- Další příkaz „burst“ je ve frontě před aktuálním dokončeným.
- Další vybraná oblast adres je stejného fyzického řádku jako současně programovaná.

Při začátku programování bude první bajt naprogramován stejně rychle jako ve standardním programovacím režimu. Zápis dalších bajtů do paměti již bude probíhat ve stejných časových intervalech až do dokončení zápisu celého segmentu. Je to způsobeno spínáním nábojové pumpy v případě příchodu nového příkazu a rozpínání nábojové pumpy při dokončení programování paměťového segmentu.

Tento programovací režim má pozitivní vliv na celkovou životnost paměti flash. Algoritmus „burst“ příkazu je navržený pro zrychlení programování viz obr. 30.



Obr. 30: Programování většího množství dat u HCS08JM - vývojový diagram

3.4.2 Flashovací funkce pro MCU MCF51JM „runOnStack“

Tato podkapitola se věnuje popisu nově vytvořených flashovacích funkcí pro 32bitové mikrokontroléry ColdFire. Základem pro tvorbu těchto nových funkcí bylo studium flashovacích funkcích pro vyšší modely procesorů ColdFire (V2), jejichž zdrojový kód byl napsán v jazyku ANSI C. Pro minimální požadavky bootloaderu na paměť, byly flashovací funkce napsány v jazyku assembler. Zaměření bylo kladeno především na použití klasického příkazu pro smazání paměti flash a také na příkaz „burst“. V následujících odstavcích se nachází popis jednotlivých funkcí i s praktickými ukázkami zdrojového kódu. Teoretická část flashování vychází ze stejných principů jako rodiny MCU HCS08, proto je zde zmíněna jen okrajově.

Příkaz zapisování sekvence příkazu

Před začátkem zapisování musí být smazány bity FPVIOL a FACCER v registru FSTAT a musí být nastaven bit FCBEF. Zapisování do paměti spočívá v třech krocích, které musí být striktně dodrženy. Postup zapisování programovací sekvence je následující:

1. Zapsání správné adresy, na kterou se má programovat.
2. Zapsání příslušného příkazu do registru FCMD.
3. Smazání bitu FCBEF v FSTAT registru a nastavení 1 bitu FCBEF pro spuštění příkazu.

Tento příkaz zapisuje blok dat na předem smazanou oblast paměti použitím programovacího algoritmu. Dva vnitřní datové registry pracují - jeden jako „buffer“ a další jako registr tak, že druhý příkaz programování s nezbytnými daty může být uložen do „bufferu“ do té doby než je první zpracován. Tato souběžná operace umožňuje časovou optimalizaci v případě, pokud je zapisováno více než jedna adresa.

1. Zapsat do paměti flash adresu místa kde se má použít příkaz programování.
2. Zapsat do registru FCMD hodnotu 0x25.
3. Smazání bitu FCBEF v FSTAT registru a nastavení 1b FCBEF pro spuštění příkazu.
4. Po té co je nastaven bit FCBEF v registru FSTAT algoritmus vstupuje znovu do kroku 1. Adresa je zvyšována automaticky vnitřním mechanismem.

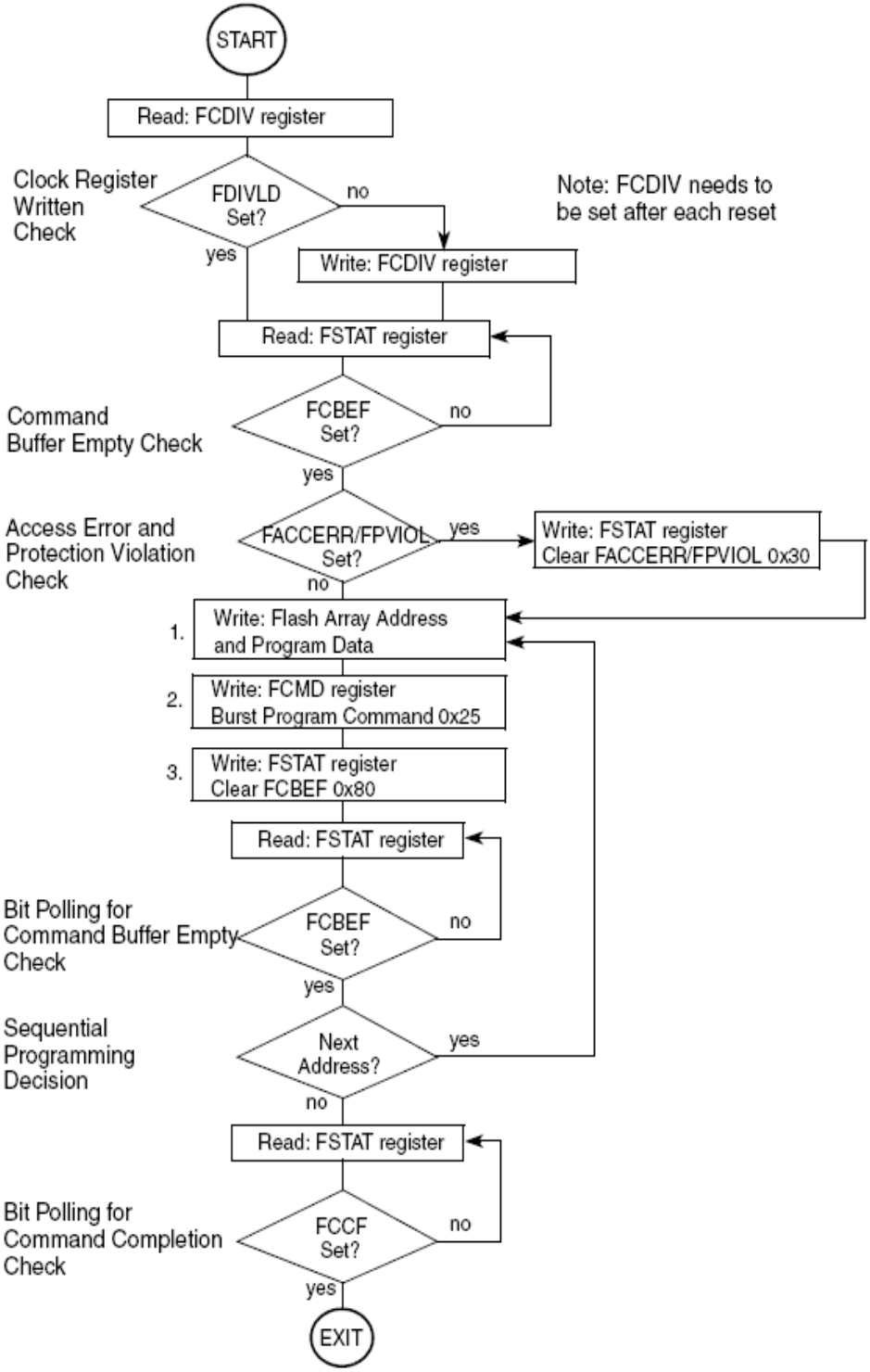
Tato procedura může být použita k programování celé flash paměti i při přeskoku hranice řádku další stránky v poli paměti flash. Jestliže při zápisu dat nastane chyba v rámci zápisu do chráněné oblasti paměti, dojde k nastavení bitu FPVIOL v registru FSTAT a příkaz se nespustí. Po dokončení jednoho programovacího cyklu se nastaví bit FCCF a dochází k ukládání nových dat do „bufferu“.

Vývojový diagram programovací procedury je zobrazen na obr. 31. V následujících části textu je uvedena část zdrojového kódu příkazu „burst“.

Ukázka zdrojového kódů flashovací funkce „burst“ v assembleru

Následující ukázka zdrojového kódu ukazuje praktický zápis programovacího příkazu „burst“ , který vychází z vývojového diagramu zobrazeném na obr. 31.

```
BURST:
        move.l  #(SpSubBurstEnd),a1 - uložení adres návěští
        move.l  #(SpSubBurst),a6
        bsr     RunOnStack
FCBEF1:
        move.l  a0,a1
        moveq   #48,d0
        move.b  d0,FSTAT
loop1:
        move.b  FSTAT,d0
        btst    #7,d0
        beq.s   loop1           - testování bitu FCBEF
        jsr     (A1)            - skok do paměti RAM na adresu v A1
        cmpi.b  #1,d6
        beq.l   ILOP1           - vykonání ilegální instrukce
        bra     SUCC
SpSubBurst:
        move.l  d2,a1           - načtení adresy
SpSubBurst1:
        move.l  (a3),(a1)       - načtení dat
        moveq   #0x25,d0        - vložení příkazu „burst“
        move.b  d0,FCMD         - do registru FCMD */
        move.l  #FSTAT,a4
        move.b  #0x80,d1        - smazání bitů FACCER a FPVIOL
        move.b  d1,(a4)
loop3:  move.b  FSTAT,d0        - jestliže je bit FCBEF = 0,
        btst    #7,d0             skoč na loop3
        beq.s   loop3
        addq.l  #4,a0           - posun na další byte
        addq.l  #4,a3
        subi.l  #4,d4
        cmpi.b  #0,d4
        bne.s   SpSubBurst1      - další sekvence k zápisu
loop4:  move.b  FSTAT,d0
        btst    #6,d0
        beq.s   loop4
SpSubBurstEnd:
        rts                     - návrat do hlavního programu
```



Obr. 31: Vývojový diagram pro příkaz „burst“ [4]

Mazání dat pomocí příkazu „sector erase“

Tato procedura je určená k smazání jednotlivých paměťových sektorů. Velikost sektoru ke smazání je 1024 bajtů. Příklad průběhu procedury mazání je zobrazen na obr. 8.

Příkaz „sector erase“ má následující sekvenci zápisu:

1. Zapsaní do paměti flash adresz místa, které má být smazáno. Adresa musí být v rozsahu 1024 bajtů sektoru, který má být smazán
2. Zapsání příkazu „secor erase “ do registru FCMD.
3. Smazání bitu FCBEF v FSTAT registru a nastavení 1 bitu FCBEF pro spuštění příkazu.

Jestliže při mazání dat nastane chyba v rámci mazání chráněné oblasti paměti, dojde k nastavení bitu FPVIOL v registru FSTAT a příkaz se nespustí. Po úspěšném dokončení příkazu „sector erase“ se nastaví bit FCCF. Vývojový diagram procedury mazání je zobrazen na obr. 32. V následujících části textu je uvedena část zdrojového kódu příkazu „sector erase“.

Ukázka zdrojového kódů flashovací funkce „erase“

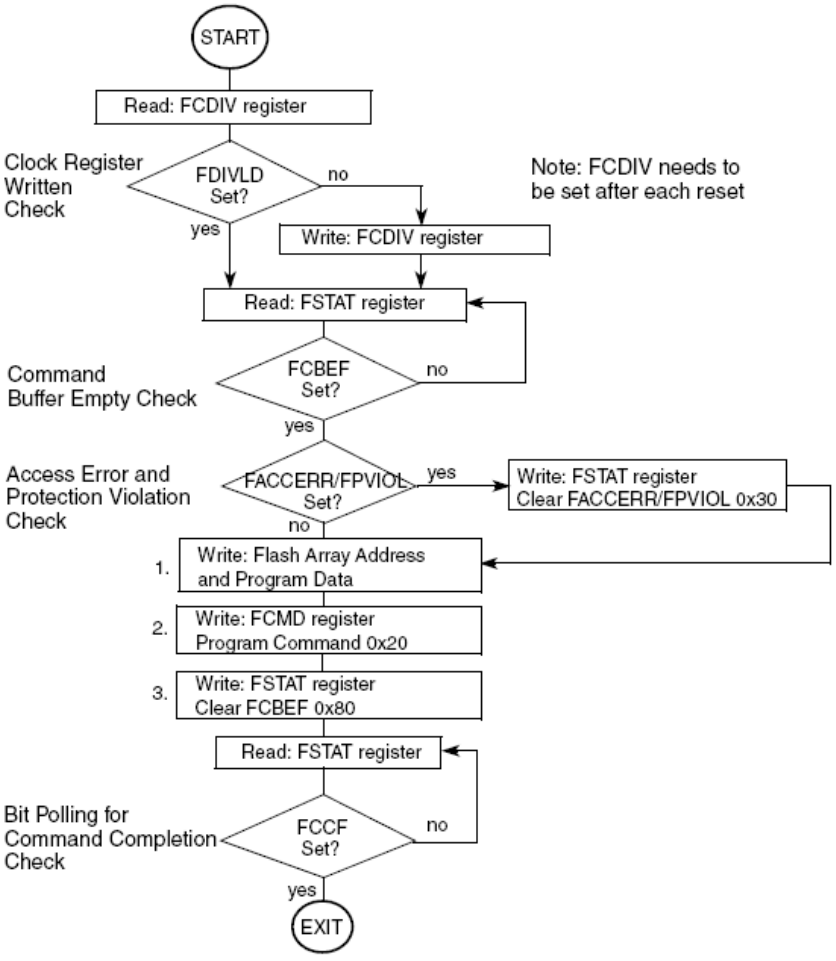
```
ERASE_COM:
            bsr      READ
            lsl.l    #8,D0
            bsr      READ
            lsl.l    #8,D0
            bsr      READ
            lsl.l    #8,D0
            bsr      READ
            move.l   d0,d1
            move.l   #(SpSubEnd),a1   - uložení poslední adresy do A1
            move.l   #(SpSub),a6      - uložení první adresy do A6
            bsr      RunOnStack       - spuštění kopírovací procedury
            move.l   a0,a1
            moveq    #30,d0           - nastavení bitu FACCER A FPVIOL
            move.b   d0,FSTAT

loop:       move.b   FSTAT,d0
            btst     #7,d0     - testování bitu FCBEF (prázdý buffer)
            beq.s    loop
            move.l   d1,a0     - vložení adresy bloku paměti do A0
            move.l   #0xFF,(a0)  -  prázdná(dummy)  data
            moveq   #0x40,d0
            move.b   d0,FCMD  - vložení příkazu do FCMD registru
            jsr     (A1)      - skok na adresu v registru A1
            bra     SUCC
```

/* Uložení počáteční adresy bloku ke smazání pomocí bitového posunu a uložení do registru D1*/

Tato část zdrojové kódu je určena k nakopírování do paměti RAM a má za úkol smazání jednoho sektoru. Bližší popis je uveden za jednotlivými řádky zdrojového kódu.

```
SpSub:
       move.l   #FSTAT,a4    - načtení adresy registru FSTAT
       move.b   #0x80,d1     - nastavení masky pro smazání bitu FCBEF
       move.b   d1,(a4)      - zapsání do registru FCBEF
loop_1:
       move.l   #FSTAT,a4    - načtení adresy registru FSTAT
       move.b   (a4),d2      - nahrání
       moveq    #0,d1        - nulování registru d1
       move.b   d2,d1
       andi.l   #0x40,d1
       beq.s    loop_1       - jestliže není nastaven bit FCCF,
                               tak skoč na loop_1
SpSubEnd:rts
```



Obr. 32: Vývojový diagram pro příkaz „sector erase“ [4]

4 Závěr

V teoretické části diplomové práce jsem podrobně popsal a na několika případech zobrazil vlastní funkce bootloaderu. Uvedl jsem některé druhy používaných bootloaderů a rozdíly mezi nimi. Je zde nahlédnuto na použité typy mikrokontrolérů a jejich vlastnosti a to zejména vlastnosti paměti flash. Stručný rozbor parametrů pamětí flash pro oba typy mikrokontroléru slouží jako tzv. náhled na problematiku. Detailnější pohled je rozebrán v kapitole praktická část.

Hlavními a nejdůležitějšími výstupy této diplomové práce jsou nově vytvořené zdrojové kódy pro bootloadery řady HCS08 a řady ColdFire (V1). Tyto zdrojové kódy, jak už je uvedeno v záhlaví této práce, jsou napsány v jazyku ANSI C. Velmi důležitým faktorem ovlivňujícím psaní celého zdrojového kódu byl parametr „velikost bootloaderu“. Zdrojový kód pro oba typy mikrokontrolérů byl tedy co nejvíce minimalizován. Celková velikost bootloaderu pro HCS08JM60 (6kB) a pro MCF51JM128 (8kB). Největším podílem na velikosti bootloaderu jsou ovladače od USB sběrnice, které zabírají cca 4kB. Rozdílné velikosti jsou následkem odlišného adresového systému a dále různými verzemi compilerů (různé instrukční sady MCU). Pro srovnání sériový (RS-232) bootloader, který jsem vytvořil pro stejný typ mikrokontroléru (MCF51JM128) měl velikost pouhé 2kB.

Dalším důležitým výstupem z této diplomové práce je vytvoření flashovacích funkcí pro mikroprocesory Cold Fire V1 v jazyku assembler. Použití těchto flashovacích funkcí není omezeno jen na použití pro bootloadery ale i pro řešení dalšího software pro tento typ mikrokontrolérů. Tyto funkce byly opatřeny hlavičkovým souborem pro kompatibilitu s jazykem C. Výsledná velikost flashovacích funkcí je 900B, což je ve srovnání s 8bitovými mikrokontroléry, kde se velikost pohybuje okolo 400-500B, výborný výsledek.

Tato práce by měla sloužit k rozvoji softwarového programování a také jako podpora všech vývojových pracovníků, pracujících s danými typy mikrokontrolérů. Vytvořením nového protokolu pro 32bitovou řadu mikrokontrolérů otevírá možnosti dalšího doplňování nových typu jednotlivých mikrokontrolérů, používaných v současnosti i v budoucnosti.

5 Seznam použitých zdrojů

[1] LAJŠNER, P. *Developer's Serial Bootloader for M68HC08 and HCS08 MCUs.* Rožnov pod Radhoštěm: Freescale Semiconductor, 2006. 52 stran.

[2] GALLOP, M., McNAMEE, J. *HCS12 Load RAM and Execute Bootloader User Guide.* Freescale Semiconductor, 2004. 16 stran.

[3] MC9S08JM60. *Datasheet for HCS08 Microcontrollers,* 2008. 386 stran.

[4] MCF51JM128RM. *MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual,* 2008. 570 stran.

[5] *USB 2.0 Specification* [online]. Usb.org, 1995 [cit. 2000-08-27]. Angičtina. Dostupný z WWW: <http://www.usb.org/developers/docs/>.

[6] JIANG, W. *USB and Using the CMX USB Stack with 9S08JM Device.* Asia & Pacific Operations Microcontroller Division, 2008. 36 stran.

[7] *Www.wikipedia.cz* [online]. 1995 [cit. 2000-01-01]. Dostupný z WWW: <[7] http://cs.wikipedia.org >.

[8] Liu, D. *USB Bootloader for MCF51JM128.* Freescale Semiconductor, 2008. 20 stran.

[9] Yang, P. *USB Bootloader for MC9S08JM60.* Freescale Semiconductor, 2008. 18 stran.

[10] DEMOJMUM. *DemoJM User Manual,* 2008. 47 stran.

Seznam symbolů, veličin a zkratek

ASCII *(American Standard Code for Information Interchange)* - americký standardní kód pro výměnu informací.

MCU *(Microcontroller unit)* - jednočipový mikropočítač.

CPU *(Central Processing Unit)* – centrální procesorová jednotka.

PC *(Personal Computer)* – osobní počítač.

USB *(Universal Serial Bus)* - univerzální sériová sběrnice, sloužící k modernímu způsobu připojení periférií k počítači. Nahrazuje dříve používané způsoby připojení (sériový a paralelní port, PS/2, GamePort apod.)

RAM *(Random-access memory)* – energeticky závislá paměť s náhodným přístupem (bez napájení ztrácí veškerý obsah informace).

ROM *(Read-Only Memory)* – energeticky nezávislá paměť, jejíž obsah nelze přepsat běžným způsobem.

RS-232 - je rozhraní pro přenos informací vytvořené původně pro komunikaci dvou zařízení do vzdálenosti 20 m. Přenos informací probíhá asynchronně, pomocí pevně nastavené přenosové rychlosti a synchronizace sestupnou hranou startovacího impulzu.

CRC *(Cyclic redundancy check)* - cyklický redundantní součet, e speciální hašovací funkce, používaná k detekci chyb během přenosu či ukládání dat.

HID *(Human Interface Device)* - je typ počítačového zařízení, který komunikuje přímo s člověkem. Jedná se rovněž o vrstvu ve struktuře ovladače USB.

CDC *(Control device class)* - řídící komunikační vrstva protokolu, implementovaná vedlejším řídícím modulem pro sériovou emulaci.

ILOP *(Illegal operation)* – ilegální instrukce mikrokontroléru způsobující reset MCU.

ANSI C - je programovací jazyk, který vyvinuli Ken Thompson a Dennis Ritchie pro potřeby operačního systému Unix. V současné době je to jeden z nejpopulárnějších jazyků, zřejmě nejčastější pro psaní systémového softwaru, ale velmi rozšířený i pro aplikace.

CAN *(Controller Area Network)* - Jedná se o sérivou datovou sběrnici vyvinutou firmou Robert Bosch GmbH. Elektrické parametry fyzického přenosu jsou specifikované normou ISO 11898. Maximální teoretická rychlost přenosu na sběrnici je 1 Mbit/s.

SPI (*Serial Peripheral Interface*) - je sériové periferní rozhraní. Používá se pro komunikaci mezi řídícími mikroprocesory a ostatními integrovanými obvody (EEPROM, A/D převodníky, displeje).

I2C *(Inter-Integrated Circuit)* - je multi-masterová počítačová sériová sběrnice vyvinutá firmou Philips, která je používána k připojování nízkorychlostních periferií k základní desce, vestavěnému systému nebo mobilnímu telefonu.

BDM *(Background debug mode)* – rozhraní pro ladění systémů používaný u novějších typů MCU.

JTAG *(Joint Test Actoin Group)* - je standard definovaný normou IEEE 1149.1, tzv. Standard Test Access Port (TAP). Jedná se o architekturu Boundary-Scan pro testování plošných spojů, programování FLASH pamětí apod.

MON08 *(Monitor 08)* - rozhraní pro ladění systémů používaný u starších typů MCU.

RISC (*Reduced Instruction Set Computer*) - označuje jednu z architektur procesorů. V překladu počítač s redukovanou instrukční sadou.

IRQ (*Interrupt ReQuest*) - označuje signál, kterým požádá zařízení (např. klávesnice, časovač atd.) o přerušení probíhajícího procesu za účelem provedení důležitější akce.

Baud - je jednotka modulační rychlosti (také znaková rychlost nebo anglicky baud rate) udávající počet změn stavu přenosového média za jednu sekundu.

MSB *(Most Significant Bit)* - se používá pro bit s nejvyšší hodnotou v binárním vyjádření čísla; v obvyklém dvojkovém zápisu jde o bit nejvíce vlevo.

FIFO (*First In, First Out*) - fronta používaná v operačních systémech pro meziprocesovou komunikaci je také nazývána roura (angl. *pipe*). Opakem fronty FIFO je zásobník (LIFO).

JMP *(Jump)* – instrukce mikrokontroléru zapříčiňující nepodmíněný skok v programu.

USB On-The-GO - je dodatkem k USB 2.0 a přidává další možnosti zařízením, které obvykle připojujeme k počítači. Mezi takováto zařízení patří např. PDA, mobily, tiskárny, přenosné disky, MP3 a jiné audio přehrávače. Všem těmto zařízením je umožněno pomocí USB OTG komunikovat přímo bez přítomnosti PC.

Obsah přiloženého CD

CD obsahuje dva hlavní adresáře. V prvním adresáři, který je pojmenován Diplomová práce, je uložen text samotné diplomové práce (DP.pdf).

Druhý adresář, který se jmenuje program, obsahuje oba vytvořené programy. V adresáři JM60 a JM128 jsou uvedeny kompletní projekty obou bootloaderů, které jsou vytvořeny pod programem CodeWarior for Microcontrollers v6.1 (jm60.mcp, jm128.mcp). Dále se zde nachází volně šiřitelná verze programu CodeWarior for Microcontrollers v6.1, která by měla sloužit k případnému otevření a prohlížení projektů.