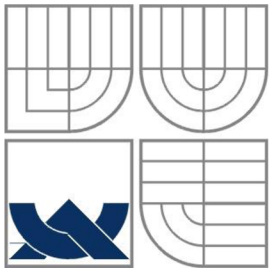
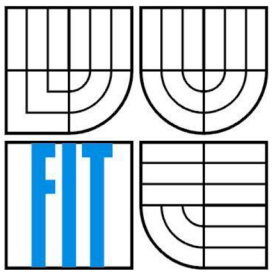


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UNIVERZÁLNÍ ZAVADĚČ PRO MIKROKONTROLÉR KINETIS K60

UNIVERSAL BOOTLOADER FOR KINETIS K60 MICROCONTROLLER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. TOMÁŠ KRŮPA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VÁCLAV ŠIMEK

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Krůpa Tomáš, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Univerzální zavaděč pro mikrokontrolér Kinetis K60**
Universal Bootloader for Kinetis K60 Microcontroller
Kategorie: Vestavěné systémy

Pokyny:

1. Podrobně prostudujte obvodové zapojení výukové platformy Minerva, kde se zaměřte na rozhraní USB, Ethernet, RS-232 a SDHC.
2. Seznamte se s mikrokontrolérem Kinetis K-60, jeho hardwarovou architekturou, principem činnosti a dostupnými vývojovými nástroji.
3. Navrhněte koncepci univerzálního zavaděče, který umožní volitelné zavedení uživatelské aplikace přes výše uvedená rozhraní a její spuštění.
4. Proveďte implementaci navrženého zavaděče a rovněž vytvořte potřebné ovladače pro PC.
5. Připravte demonstrační aplikaci, která prokáže funkčnost navrženého řešení.
6. Zhodnoťte dosažené výsledky a pokuste se navrhnout případná vylepšení či rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šimek Václav, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Cílem práce bylo vytvoření zavaděče pro mikrokontrolér Kinetis K60 s jádrem ARM Cortex-M4, který by usnadnil zavádění aplikací na vývojové platformě Minerva osazené tímto mikrokontrolérem. Na základě průzkumu dostupných zavaděčů byl proveden návrh nového zavaděče a souvisejícího komunikačního protokolu, který je využit pro přenos souborových dat a konfiguračních informací napříč všemi podporovanými komunikačními rozhraními – USB, SDHC, Ethernet a RS-232. Na straně PC proběhl vývoj aplikační knihovny pro práci se zavaděčem a dvou pomocných utilit s grafickým a konzolovým rozhraním. Výsledný systém je připraven na budoucí možné rozšiřování i pro ostré nasazení do praxe.

Abstract

The scope of this work was to design a bootloader for Kinetis K60, an ARM Cortex-M4 microcontroller. The bootloader should simplify the process of loading user applications to the flash memory of the microcontroller on the Minerva development kit. Based on a quick research of existing bootloader solutions, new bootloader was designed together with communication protocol that would suit the needs for transfer of both the file data and configuration information through all the supported communication interfaces – USB, SDHC, Ethernet and RS-232. On the PC side, development of bootloader's application library and two supporting utilities with graphical and command line interfaces was made. The system can be used out of the box as well as it can be further extended in the future.

Klíčová slova

zavaděč, Minerva, Kinetis, K60DN512, Cortex-M4, ARM, USB CDC, Ethernet, SDHC, RS-232, FatFs, Processor Expert, KDS, komunikační protokol, OSBDM

Keywords

bootloader, Minerva, Kinetis, K60DN512, Cortex-M4, ARM, USB CDC, Ethernet, SDHC, RS-232, FatFs, Processor Expert, KDS, communication's protocol, OSBDM

Citace

KRŮPA, Tomáš. *Univerzální zavaděč pro mikrokontrolér Kinetis K60*. Brno, 2015. 69 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Šimek Václav.

Univerzální zavaděč pro mikrokontrolér Kinetis K60

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Václava Šimka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Krůpa
19.5.2016

Poděkování

Chtěl bych poděkovat panu Ing. Václavu Šimkovi za rady, náměty a kritické podněty při řešení diplomové práce.

Obsah

1 Úvod

Každý počítačový systém, předtím než začne vykonávat uživatelský program, musí projít fází inicializace. Během inicializace se provádí mimo jiné zavádění uživatelského programu do paměti a jeho spuštění. A právě k tomuto účelu slouží software, který se nazývá zavaděč. V angličtině se pak setkáváme s ekvivalentním pojmem *bootloader*.

Z pohledu běžného uživatele je zavaděč ve většině případů naprosto transparentní a často ani nemusí vědět, že nějaký takový software ve svém zařízení má. Avšak jen díky němu zařízení funguje podle očekávání, protože zavaděč je zodpovědný za nahrání a spouštění uživatelského programu. Z pohledu výrobce počítačového systému pak zavaděč usnadňuje jeho programování během výroby a provádění aktualizací programového vybavení počítačového systému v terénu, protože eliminuje potřebu použití externích programovacích zařízení. V závislosti na složitosti systému bývá zavaděč různě sofistikovaný a u systémů na bázi jednoduchých mikrokontrolérů nemusí být přítomen vůbec. Složitější systémy používají zavaděče k počáteční inicializaci hardware, nastavení prostředí pro běh programů a především k načtení uživatelského programu z externího paměťového úložiště (pevný disk, flash paměť) do operační paměti a k přesměrování toku řízení do tohoto programu. Tím může být např. jádro operačního systému, nebo specializované vestavěné aplikace.

Tato práce se dále zaměřuje pouze na zavaděče používané v kombinaci s mikrokontroléry ve vestavěných systémech. V následujících deseti kapitolách popisuje vývoj nového zavaděče a souvisejících nástrojů, včetně příkladů jejich použití s konkrétní vestavěnou aplikací. Mezi hlavní požadavky kladené na nový zavaděč patří vysoká univerzalita vyznačující se širokou škálou podporovaných komunikačních rozhraní a také snadná použitelnost. Vývoj zavaděče probíhal na výukové platformě Minerva, která je osazena cílovým mikrokontrolérem Kinetis K60 (dále jen MK60) s jádrem ARM Cortex-M4 od firmy Freescale (nynější firma NXP).

Druhá kapitola obecně přibližuje způsob použití zavaděče na platformě ARM. Je popsána inicializace mikrokontroléru (dále jen MCU) následovaná detailnějším vysvětlením principu funkce zavaděče. Dále jsou uvedeny dva příklady existujících zavaděčů s vyznačenými klady a zápory vzhledem k požadavkům kladeným na nový zavaděč. Třetí kapitola podrobněji představuje výukovou platformu Minerva. Je zde analyzováno obvodové zapojení platformy s důrazem na ty periferie, které by měly být podporovány v novém zavaděči. Také je nastíněna možnost přístupu k těmto periferiím ze software.

Od kapitoly páté dále se nachází popis vlastního řešení. Nejprve je popsána softwarová architektura nového zavaděče spolu s komunikačním protokolem navrženým pro přenos dat mezi aplikací na straně PC a zavaděčem na straně MCU. Podrobně je rozebráno spouštění, běh a ukončení činnosti zavaděče. Šestá kapitola představuje knihovnu pro komunikaci se zavaděčem, která je dostupná aplikacím na straně PC. Je zde popsán návrh knihovny s vysvětlením funkce

nejdůležitějších objektových tříd a možnosti využití knihovny v externích aplikacích. Ty jsou představeny v kapitole sedmé a osmé. Jedná se o dva nástroje s grafickým a konzolovým rozhraním, které slouží ke konfiguraci zavaděče a nahrávání uživatelských aplikací do MCU z prostředí operačních systémů Windows a Linux. Konkrétní příklad použití zavaděče v kombinaci s vytvořenými utilitami se nachází v kapitole deváté. Pro ukázání funkčnosti zavaděče byla zvolna existující demonstrační aplikace WEB HVAC pro platformu Minerva. V poslední kapitole jsou shrnuty dosažené výsledky s vyznačenými rozšířeními, které byly implementovány nad rámec zadání. Rovněž jsou naznačena další možná vylepšení a rozšíření.

Tato diplomová práce navazuje na stejnojmenně nazvaný semestrální projekt, který byl vypracován na přelomu roku 2015/2016. Ze semestrálního projektu jsou s úpravami a rozšířeními převzaty kapitoly 1, 2, 3, 4 a 5. Kompletně byla převzata implementace zavaděče, která byla pro účely diplomové práce pouze rozšířena o nové funkce a byla lépe odladěna. Ostatní části projektu jsou výstupem diplomové práce.

2 Zavaděče pro ARM

V této kapitole se nachází informace týkající se inicializace a zavádění uživatelského programu na mikrokontrolérech z rodiny ARM, do které spadá i cílový mikrokontrolér MK60. Z hlediska MCU se zavaděč nijak neliší od jakéhokoliv jiného programu pro vestavěné zařízení – před spuštěním zavaděče se MCU musí nacházet v definovaném stavu.

2.1 Inicializace mikrokontroléru

Před provedením první instrukce programu se mikrokontrolér nachází vždy ve výchozím stavu, který je popsán v manuálu. Všechny registry jádra a připojených periferií obsahují definované hodnoty, většina periferií je zakázána a má zcela odpojené napájení, všechny všeobecně použitelné piny mají nastaven vstupní směr a mikrokontrolér bývá taktován z interního hodinového oscilátoru realizovaného obvykle pomocí několika RC článků a operačního zesilovače (tzv. IRC obvod). Inicializace mikrokontroléru MK60 probíhá podle následujícího postupu [1]:

1) Vnitřní resetovací logika drží celý mikrokontrolér v resetu, dokud nejsou stabilizována všechna napájecí napětí. Tento krok se provádí pouze při resetu po připojení napájení (tzv. power-on reset), jelikož v případě vnitřně generovaného resetu (programový reset, reset generovaný hlídacím obvodem…) jsou již napětí stabilní.
2) V dalším kroku je spuštěn vnitřní hodinový generátor (MCG) a povoleny hodinové rozvody pro jádro, systémové sběrnice a flash kontrolér. Výchozí takt jádra je odvozen od vnitřního 32kHz IRC oscilátoru, který je použit jako reference pro obvod frekvenčního závěsu s násobícím faktorem 640. Výsledná frekvence jádra je tedy 20,97 MHz.
3) Je provedena inicializace flash kontroléru, během které je jádro MCU stále drženo v resetu. Po dokončení inicializace je buď spuštěna inicializace jádra, nebo je vyvolán speciální režim běhu nazývaný EZ-Mode, který umožňuje pomocí rozhraní SPI plný přístup k vnitřní flash paměti za účelem mazání, programování a čtení.
4) Ve většině případů však bude spuštěna inicializace jádra, během které je nastaven registr návratové adresy (LR) na hodnotu 0xFFFFFFFF a dále registry programového čítače (PC) a ukazatele na vrchol zásobníku (SP) podle hodnot v tabulce vektorů, která je umístěna ve flash paměti na adrese 0. Následně započne vykonávání programu od adresy v registru PC.

2.2 Činnost zavaděče

Před spuštěním hlavní smyčky zavaděče je potřeba provést některé inicializace na aplikační úrovni, mezi které patří:

- Inicializace vnitřních volatilních pamětí (RAM) z nevolatilních pamětí (flash). Tento krok je potřeba k nastavení hodnot inicializovaných proměnných zavaděče. Úsek programu, který toto zajistí, bývá obvykle automaticky vložen kompilátorem, nebo se jedná o část tzv. startovacího kódu.
- Nastavení adresy k tabulce vektorů přerušení. Zpravidla tento krok není nutné dělat, protože lze využít standardní umístění tabulky vektorů na adrese 0 ve flash paměti. Bude jej však nutné provést, pokud bude zavaděč spouštět jinou aplikaci s vlastní tabulkou přerušovacích vektorů, nebo bude-li zavaděč měnit svoji tabulku přerušovacích vektorů za běhu.
- Provedení dodatečné inicializace hardware, ať už se jedná o externí či interní periferie MCU. Tento krok zahrnuje například inicializaci vstupně-výstupních pinů, nastavení hodinového subsystému a přednastavení jednotky správy paměti (MMU).

Prvním krokem každého zavaděče po provedení základní inicializace je rozhodnutí, zda má spustit uživatelský program či nikoliv. Kritéria, kterými se zavaděč řídí, závisejí jednak na jeho složitosti a jednak na požadavcích uživatele. Nepožaduje-li uživatel od zavaděče žádnou interakci, pak zavaděč zpravidla pouze zjišťuje, zda je v zařízení nahrán platný program a pokud ano, tak mu předá řízení a tím jeho činnost končí. Pokud žádný uživatelský program není dostupný, či je poškozený, nebo v případě, kdy je explicitně vyžadován běh zavaděče, pak se namísto uživatelského programu spustí samotný zavaděč, se kterým může uživatel komunikovat a provádět další akce.

Pokud se již zavaděč kompletně spustil, pak je to obvykle z důvodu, že se v zařízení nenachází platný program. Zavaděč v tomto případě slouží k jeho nahrání. Nahrávání probíhá pomocí některého z podporovaných komunikačních kanálů, mezi které často patří jeden nebo více sériových kanálů USB, RS-232, SPI, I^2C, UART nebo CAN. Zavaděči se zasílá buď přímo binární obraz aplikace (nejčastěji soubor s příponou .bin), nebo soubor obsahující popis obsahu paměti v některém z podporovaných formátů, např. Intel HEX (.hex), Motorola S-Record (.s19) apod. Nahrávání binárních obrazů do flash paměti přestavuje jednodušší způsob zavádění aplikace do zařízení, neboť zavaděč nemusí implementovat překladač do binárního formátu. Na druhou stranu mohou mít binární soubory větší velikost oproti jiným formátům a důsledkem toho může být delší doba přenosu a nahrání uživatelské aplikace do cílového zařízení.

Druhým častým důvodem spuštění zavaděče je aktualizace stávajícího uživatelského programu. Nahrávání souboru s aplikací pak probíhá stejně jako v prvním případě. U vestavěných zařízení spadajících do kategorie internetu věcí (Internet of Things) není aktualizace firmware nic výjimečného a zavaděče s podporou nejrůznějších rozhraní zde nacházejí své uplatnění.

Po nahrání nového firmware do zařízení zavaděč zkontroluje správnost jeho uložení ve vnitřní paměti pomocí nějaké formy zabezpečení dat, ať už pomocí CRC součtu nebo pomocí hashovací funkce (MD5, SHA-1…). Vypočítaná kontrolní data si zavaděč uloží do své perzistentní paměti, aby mohl před příštím spuštěním nahrané aplikace provést kontrolu její integrity. Kontroly se nemusejí omezit pouze na porovnání kontrolních dat – je možné také vyhledávat definované vzory v binárních souborech a tím ověřit platnost nahraného programu.

Následně je přesměrován tok programu do uživatelského programu, čemuž předchází přemapování tabulky vektorů. Přemapování je téměř vždy nutné, protože uživatelský program používá své vlastní obslužné rutiny přerušení (ISR), které se po zavedení programu nacházejí na jiných adresách v paměti, než ty, které využívá zavaděč.

2.3 Kategorie zavaděčů

Tato podkapitola obsahuje stručný přehled různých kategorií zavaděčů.

2.3.1 Zavaděč v ROM

Tyto zavaděče umisťují do některých svých mikrokontrolérů přímo jejich výrobci a nacházejí se v části paměti určené pouze pro čtení (ROM nebo flash). Jedná se tedy o dodávané softwarové vybavení, které může uživatel využít, pokud potřebuje ve svém zařízení využívat funkci zavádění systému z různých zdrojů, ale zároveň nechce obětovat čas a prostředky pro tvorbu vlastního zavaděče.

Výhodou tohoto řešení je fakt, že se zavaděč nachází v části paměťového prostoru, který není jinak uživateli dostupný. Tím pádem jsou mu veškeré paměťové zdroje MCU stále dostupné v celé své kapacitě. Nevýhodou je nutnost počítat s jeho použitím již na úrovni návrhu hardware, neboť tyto zavaděče často využívají pouze některá předem definovaná komunikační rozhraní. Může se tak snadno stát, že se zavaděč stane nepoužitelným, protože jeho jediný podporovaný UART je připojen na vstupně-výstupní piny, jejichž funkci by bylo v daném návrhu potřeba využít jinak. V tomto případě tedy platí, že čím je zavaděč univerzálnější, tím menší je pravděpodobnost takové kolize.

2.3.2 Jednorázový zavaděč

Tento zavaděč, též zvaný *flashloader*, je specifický tím, že je určen pouze pro jedno použití. Zavaděč je nejprve umístěn v nevolatilní paměti mikrokontroléru. Po připojení napájení se zavaděč zkopíruje do volatilní paměti RAM, odkud se následně spustí a provede stažení uživatelského programu pomocí zvoleného komunikačního rozhraní do té samé části nevolatilní paměti, ve které byl dříve umístěný, čímž sám sebe přepíše. Zavaděč pak může spustit uživatelskou aplikaci, která postupnými zápisy do RAM obraz zavaděče v ní umístěný přepíše a tím se zavaděč z MCU definitivně vymaže.

Výhodou tohoto způsobu je snadné programování vnitřní paměti mikrokontrolérů – výrobci vestavěných zařízení si mohou nechat dodat mikrokontroléry s tímto typem zavaděče a nahrání vlastních programů provádět jednorázově přímo na výrobní lince pomocí dostupných rozhraní. Nevýhodou je znemožnění pozdější změny uživatelského programu (vyjma případu, kdy nahrávaný obraz aplikačního programu v sobě také obsahuje nějaký zavaděč).

2.3.3 Samostatný zavaděč

Tento typ zavaděče je nejuniverzálnější, protože se nachází v uživatelské části nevolatilní paměti MCU. Může se jednat o zavaděč dodávaný výrobcem MCU nebo může být na míru vytvořen samotným uživatelem. Díky tomu je možné jej optimalizovat s ohledem na velikost binárního obrazu a na podporovaná komunikační rozhraní.

Z toho vyplývá výhoda plné kontroly nad funkcemi zavaděče. Nevýhodou je umístění v uživatelské části nevolatilní paměti, kterou pak zavaděč sdílí s dalšími uživatelskými programy. Tím se zmenšuje efektivní využitelná velikost této paměti pro nahrané programy. Ve většině případů to však nebývá závažný problém, protože zavaděče na mikrokontrolérech zabírají několik málo desítek kilobajtů a vhodnou konfigurací zavaděče či výběrem mikrokontroléru s dostatečně velkou kapacitou paměti lze tento nedostatek snadno obejít.

2.4 Existující zavaděče pro Kinetis

V této kapitole jsou představeny dva existující zavaděče, které jsou volně dostupné pro mikrokontrolér MK60.

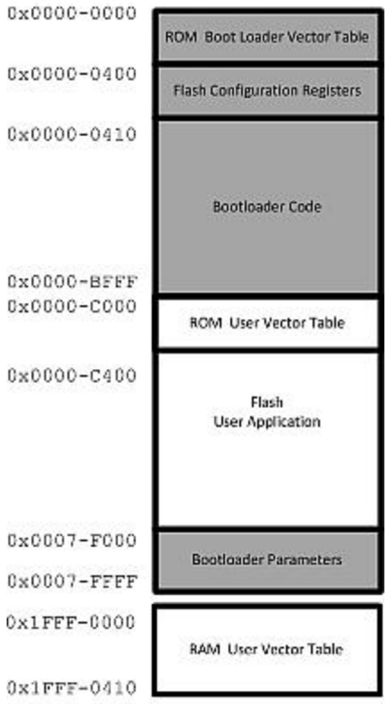
2.4.1 Ethernet bootloader

Tento zavaděč vyvinula firma Freescale pro své mikrokontroléry řady Kinetis a některé modely řady ColdFire. Zavaděč podporuje rozhraní Ethernet pro přenos uživatelského programu a asynchronní sériové rozhraní (UART) pro konfiguraci. Přenos uživatelského programu do zařízení probíhá pomocí protokolu TFTP. Pro snadnější operabilitu v různých sítích disponuje zavaděč DHCP klientem [2].

Tento zavaděč spadá do kategorie samostatných zavaděčů. Na obrázku 2.1 je vidět rozdělení flash paměti při použití s mikrokontrolérem MK60. Pro zavaděč je vyhrazena oblast o velikosti 48 KB nacházející se na začátku flash paměti a dále poslední sektor flash paměti o velikosti 4 KB pro uložení parametrů a konfigurace zavaděče. Zbytek paměti je dostupný uživatelskému programu.

Po připojení napájení zavaděč načte z konfigurační oblasti požadovaný režim běhu. Jsou podporovány 3 režimy:

1) Režim GO vede na spuštění uživatelského programu.



2.1 Rozdělení flash paměti Ethernetového zavaděče s mikrokontrolérem Kinetis MK60N512. Převzato z [2].

2) Režim STOP vede na spuštění zavaděče v uživatelském režimu. Tento režim je zároveň výchozím a je možné v něm ovládat a měnit konfiguraci pomocí terminálu dostupného skrze sériové rozhraní UART.
3) Režim SCRIPT vede na spuštění zavaděče ve skriptovacím režimu. Podle skriptu dříve zapsaného pomocí uživatelského režimu provede zavaděč požadované akce. Příklad skriptu může být následující:

   „dhcp; erase all; tftp; set boot go; save; go“

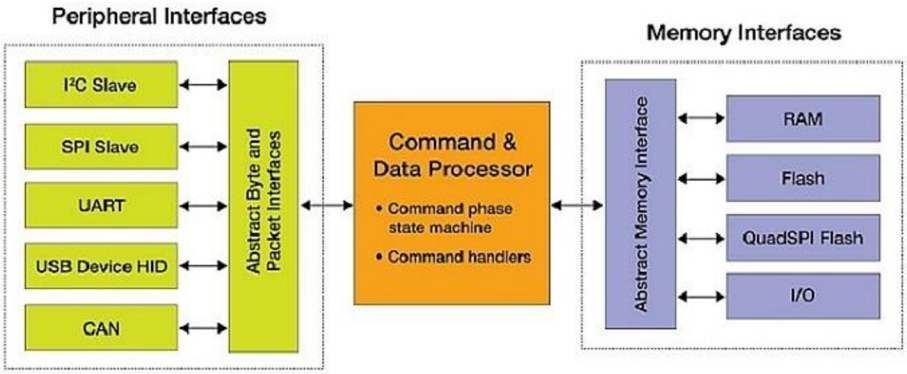
   Tento skript provede získání IP adresy z DHCP, následně vymaže uživatelskou část flash paměti a z nastaveného TFTP serveru stáhne a nahraje nový obraz uživatelského programu. Dále nastaví režim na GO, uloží konfiguraci a spustí právě nahranou aplikaci.

Největší výhodou tohoto přístupu je používání skriptu v kombinaci s TFTP serverem. Nakonfigurování zařízení proběhne pouze jednou a o všechno ostatní už se postará samotný zavaděč. Konfigurace zavaděče je taktéž přístupná uživatelskému programu, takže lze snadno provádět jeho automatické aktualizace.

Jako nevýhoda se jeví konfigurace zavaděče pouze pomocí sériové linky, protože vyžaduje použití přídavného propojení (ať už USB kabel nebo převodník UART-USB) v případě jakéhokoliv rekonfigurování. V tomto ohledu by se nabízelo lepší řešení v podobě telnet terminálu dostupného pomocí Ethernetového rozhraní.

2.4.2 Kinetis bootloader

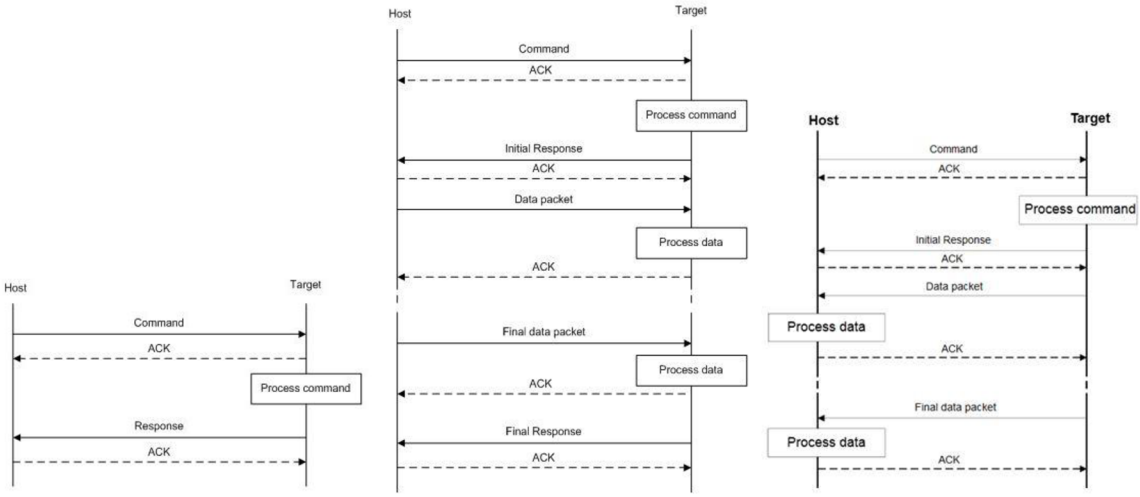
Kinetis bootloader je konfigurovatelný programovací nástroj, který pracuje skrze sériovou linku na mikrokontrolérech Kinetis firmy Freescale. Umožňuje rychlé a jednoduché programování mikrokontrolérů Kinetis po celou dobu jejich životnosti, a to i ve fázích vývoje a výroby zařízení. Zavaděč je dostupný dvěma způsoby – formou zdrojových kódů s možností konfigurace při sestavování a dále jako předprogramovaný v ROM nebo ve flash paměti podle modelu mikrokontroléru. Na straně hosta jsou dostupné nástroje příkazové řádky a také grafické nástroje pro komunikaci se zavaděčem. Uživatel může použít tyto nástroje k nahrání/stažení uživatelského kódu do/ze zařízení [3].



2.2 Blokové schéma Kinetis zavaděče. Převzato z http://cache.nxp.com.

Tento zavaděč spadá do všech tří kategorií zavaděčů definovaných výše. Na obrázku 2.2 je vidět základní funkční schéma zavaděče. Za zdůraznění stojí především jeho univerzálnost vyznačující se širokým spektrem podporovaných komunikačních rozhraní, které je možné uživatelsky povolovat či zakazovat. Po spuštění zavaděče probíhá automatická detekce komunikace na vybraných rozhraních, přičemž po navázání spojení na některém rozhraní se ostatní deaktivují. Podporovaná rozhraní lze povolovat pomocí definic ve zdrojovém kódu.

Na všech rozhraních je podporován jednotný binární komunikační protokol založený na přenosu paketů. Protokol rozlišuje několik typů přenosů – příkaz bez datové části, příkaz s datovou částí od hosta k zařízení a příkaz s datovou částí od zařízení k hostu. Tyto přenosy jsou znázorněny na obrázku 2.3.



2.3 Schéma komunikace mezi zavaděčem Kinetis (Target) a programovacím nástrojem (Host). Vlevo je přenos samostatného příkazu, uprostřed je přenos s datovou částí od hosta k zavaděči a vpravo je přenos s datovou částí směrem od zavaděče k hostovi. Zdroj [3].

Zavaděč disponuje několika pokročilými funkcemi. Mezi ně patří zabezpečení flash paměti proti čtení pomocí hesla, zpřístupnění zabezpečené paměti zadáním hesla nebo vymazáním obsahu paměti, vyčítání parametrů flash paměti daného procesoru (velikost stránky, velikost sektoru…) či spuštění zavaděče přímo z uživatelského programu.

Konfigurace zavaděče se nachází v uživatelské části flash paměti. Jedná se o 48 bajtů dat umístěných s offsetem 960 bajtů od tabulky přerušovacích vektorů uživatelského programu. Zavaděč kontroluje platnost konfigurace pomocí detekce definované sekvence bajtů na jejím začátku. Uživatelský program musí být sestaven tak, aby tuto oblast nevyužíval pro svůj kódový segment.

Ačkoliv je zavaděč napsán v jazyce C, návrh programu je silně inspirován objektově orientovaným programováním. To se vyznačuje především používáním jednotných rozhraní pro komunikaci mezi oddělenými částmi aplikace (jádro zavaděče, ovladače periferií, ovladače paměti). Návrh se taktéž vyznačuje snadnou rozšiřitelností a udržovatelností, zdrojové kódy i použité protokoly jsou velmi dobře okomentovány.

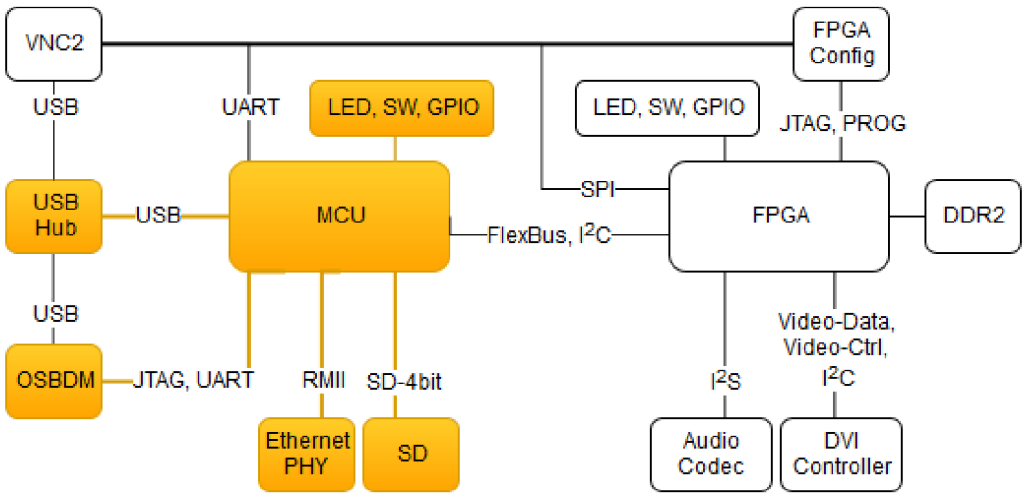
Nalézt nějaké zásadní nevýhody je v tomto případě poměrně obtížné. Používání objektových rysů v jazyce C však implikuje mírně složitější kód. Drobným problémem je také neúplná podpora pro všechny mikrokontroléry řady Kinetis – model Kinetis K60 se mezi podporovanými aktuálně nenachází (04/2016). Vyváženo je to alespoň tím, že manuál poskytuje podrobný návod na přenos zavaděče na libovolný mikrokontrolér řady Kinetis.

3 Platforma Minerva

V této kapitole se nachází stručný popis platformy, jenž disponuje potřebným mikrokontrolérem MK60, pro který má být primárně vyvinut nový univerzální zavaděč. Je popsáno schéma platformy, propojení a způsob použití jednotlivých periferií/komponent s důrazem na ty, které mají být použity během procesu zavádění nového programu do mikrokontroléru. V textu této kapitoly jsou dále naznačeny vazby mezi těmito komponenty a navrhovaným zavaděčem.

3.1 Popis zapojení

Platforma Minerva (obr. 3.1) se skládá z dílčích komponent a integrovaných obvodů umístěných na jediném plošném spoji. Platforma má charakter prototypu vestavěného systému, jelikož obsahuje výkonný mikrokontrolér Freescale Kinetis MK60DN512VMD10 v 144pinovém pouzdře MAPBGA a další specializovaný hardware reprezentovaný integrovanými obvody zajišťujícími multimediální funkce skrze reprogramovatelnou hradlovou logiku FPGA Xilinx Spartan 6 (XC6SLX9) v 200pinovém pouzdře CSG324. Přítomnost hradlového pole umožňuje výrazně rozšířit schopnosti platformy, neboť může plnit funkci téměř libovolného číslicového obvodu. Lze jej použít například k akceleraci algoritmů běžících v MCU, nebo pro komunikaci s externími specializovanými obvody připojenými k FPGA pomocí rozšiřujících portů.



3.1 Blokové schéma platformy Minerva. Oranžově zvýrazněné části jsou zajímavé z hlediska zvažovaných funkcionalit nového zavaděče.

Kromě MCU a FPGA se na plošném spoji nachází řada podpůrných obvodů. Prvním z nich je převodník USB/COM FTDI Vinculum-II (VNC2), který zpřístupňuje sériová rozhraní MCU a FPGA

prostřednictvím sběrnice USB. Komunikace s použitím těchto převodníků je velmi snadná, jelikož ovladače pro tyto převodníky jsou přímo součástí, nebo jsou snadno dostupné pro všechny dnes běžně rozšířené operační systémy.

Dalším významným celkem integrovaným na plošném spoji je ladící rozhraní OSBDM s mikrokontrolérem Freescale HCS08 (MC9S08JM60). Toto rozhraní slouží k programování a krokování aplikací na mikrokontroléru Kinetis. Propojení mezi HCS08 a MK60 je realizováno pomocí 5pinového rozhraní JTAG. Rozhraní také zpřístupňuje druhý asynchronní sériový kanál UART mikrokontroléru MK60 pomocí sběrnice USB. Podrobnější informace o sériovém rozhraní se nacházejí v kapitole 3.5.

Propojení všech USB zařízení na vývojové desce zajišťuje USB Hub Texas Instruments TUSB2046B podporující Full-Speed USB specifikaci s rychlostí až 12Mbit/s. Díky němu je možné platformu připojit k PC pomocí jediného USB kabelu a zároveň komunikovat s jednotlivými obvody na desce.

Připojení vývojové desky k síti Ethernet zajišťuje na fyzické vrstvě obvod LAN8720. Jedná se o řadič Ethernetového portu, který na jedné straně disponuje konektorem RJ45 pro připojení Ethernetového kabelu a na druhé straně rozhraním RMII (Reduced Media Independent Interface) pro přenos paketových dat a rozhraním SMI (Serial Management Interface) pro přenos řídících informací z a do řadiče MAC (Medium Access Controller), který je integrován v MCU. Další informace k Ethernetovému rozhraní se nacházejí v kapitole 3.4.

Rozšířit kapacitu paměti je možné formou výměnných SD karet. K dispozici je standardní konektor pro SD karty plné velikosti s detekcí přítomnosti karty ve slotu a informací o zapnuté ochraně zápisu. Slot je přímo připojen k SDHC řadiči v MCU, který obstarává veškerou komunikaci. Podrobnější popis funkce tohoto rozhraní se nachází v kapitole 3.6.

Na desce se nachází také tlačítka a LED diody připojené buď k MCU nebo k FPGA. Tyto součástky budou využity v zavaděči pro interakci s uživatelem. Další detaily týkající se propojení jednotlivých komponent nejsou podstatné z hlediska zvažovaného zavaděče a lze je dohledat v podrobném schématu platformy Minerva [4].

3.2 Mikrokontrolér Kinetis K60

Mikrokontrolér MK60DN512VMD10 z řady Kinetis K60 je založen na jádře ARM Cortex-M4, které je primárně určeno pro použití ve vestavěných zařízeních. Jedná se tedy o tzv. System on Chip (SoC), který se vyznačuje především širokou škálou přidaných periferií a nízkou spotřebou. Mezi hlavní přednosti tohoto MCU patří integrovaný MAC řadič pro Fast Ethernet s podporou časové synchronizace dle standardu IEEE®1588 a USB 2.0 Full-Speed (12 Mb/s) řadič s podporou funkce On-The-Go (OTG), díky které může MCU na sběrnici USB vystupovat buď jako klientské USB zařízení nebo jako USB host. Samozřejmostí jsou integrované řadiče běžných sériových rozhraní SPI,

I^2C a UART, dále periferie pro zpracování analogového signálu (ADC, DAC) a časovače různého zaměření (Watchdog, (Low-Power) FlexTimer, SysTick, RTC).

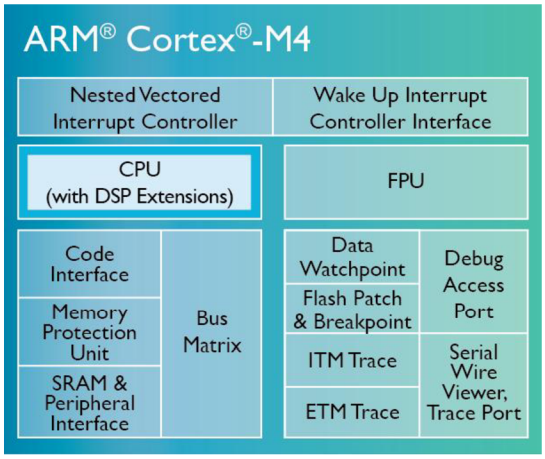
Jádro MCU může pracovat na frekvenci až 100 MHz, přičemž ostatní komponenty na čipu pracují na různých nižších frekvencích. Hlavní hodinový takt je odvozen buď přímo z interního IRC oscilátoru, nebo je možné dosáhnout přesnějšího taktu pomocí modulu frekvenčního závěsu (FLL) či fázového závěsu (PLL) synchronizovaného externím krystalem, který je na platformě Minerva osazen s frekvencí 50 MHz. Nastavení hodinových signálů jednotlivých periferií MCU je v zavaděči nutné pouze v případě, je-li požadováno plnohodnotné spuštění zavaděče. Pokud zavaděč pouze kontroluje přítomnost uživatelského programu, může tak učinit s výchozím nastavením hodinových signálů a jejich další konfiguraci přenechá až uživatelskému programu.

Interní flash paměť má velikost 512 KB a je rozdělena na dvě stejně velké poloviny. Jelikož MCU umožňuje provádění programu přímo z flash paměti, je zavedeno omezení na zápis pouze do té poloviny paměti, ze které se neprovádí kód. Pokud je potřeba zapisovat do obou polovin paměti, pak je nutné provádění kódu před zápisem dočasně pozastavit, provést zápis dat a poté lze pokračovat ve vykonávání od místa přerušení. Pozastavení může být realizováno čekací rutinou vykonávanou z RAM paměti, nebo umístěnou v druhé polovině flash paměti.

Interní paměť RAM má kapacitu 128 KB, což je pro zavaděč naprosto dostačující kapacita i z toho důvodu, že se o tuto paměť nemusí dělit s uživatelským programem, kterému ji celou po svém ukončení přenechá. Paměť bude zavaděčem využita k uložení programového zásobníku, pomocných proměnných a paměťových prostorů pro výměnu dat na různých rozhraních.

ARM Cortex-M4

Jádro ARM Cortex-M4 je postaveno na harvardské architektuře ARMv7-M a bylo navrženo především pro digitální zpracování signálů. Nepřekvapí přítomnost pokročilých DSP operací, jako je provádění operace Multiply and Accumulate (MAC) v jednom taktu, podpora SIMD instrukcí a volitelně dostupná FPU jednotka (nepřítomná na MK60) [5].



3.2 Blokové schéma jádra Cortex-M4. Zdroj [5]

Pro ladění aplikací za běhu se v jádře nachází podpora v podobě rozhraní Debug Access Port, který zpřístupňuje procesorové registry a obsah paměti externímu ladícímu nástroji (OSBDM) pomocí rozhraní JTAG nebo SWD. Díky rozhraní Debug Access Port je možné spustit vykonávání programu v ladícím režimu MCU, který umožňuje sledovat obsah paměti a zastavovat chod programu na definovaných instrukcích či při zápisu různých hodnot na specifikované adresy apod. Pro efektivní sledování toku programu lze využít modul Instrumentation Trace Macrocell (ITM), který — zjednodušeně vzato — funguje jako pomocná asynchronní sériová linka pro přenos trasovacích dat a může být použit pro přenos formátovaného textu vytvořeného pomocí funkce *printf* mezi jádrem MCU a terminálem na PC.

3.3 Rozhraní USB

Z pohledu USB sběrnice se platforma Minerva tváří jako několik samostatných USB zařízení, které jsou propojeny USB rozbočovačem. Jedná se rozhraní OSBDM, převodník VNC2 a mikrokontrolér MK60. Platforma disponuje dvěma USB konektory. První typu USB-B slouží k připojení USB rozbočovače směrem k USB hostu. Druhý typu USB-A je připojen k obvodu VNC2 a jeho využití záleží na programu nahraném v tomto obvodu. Totéž platí o mikrokontroléru MK60, který bude na sběrnici USB viditelný pouze v případě, bude-li mít aktivovánu USB periferii a nahrán příslušný firmware.

Základem USB řadiče v mikrokontroléru MK60 je obvod zvaný SIE (Serial Interface Engine), který je navržen tak, aby abstrahoval řízení signálů na sběrnici od samotného přenosu datových paketů. Z pohledu programátorského modelu se USB řadič jeví jako sada stavových registrů, řídicích registrů a datových struktur v paměti.

Data jsou pak mezi ovladačem zařízení na straně hosta a obslužnými rutinami na straně zařízení přenášena v rourách [6]. Roura je na straně hosta i zařízení identifikována pomocí tzv. endpointu (EP), což je fyzická část USB řadiče, která se stará o přenos dat zvoleného USB protokolu. Komunikace na EP může probíhat vstupním, výstupním nebo oběma směry zároveň v závislosti na vybavenosti USB řadiče. Speciální význam má EP0, který musí být obousměrný a pomocí kterého probíhá enumerace a konfigurace zařízení na sběrnici. USB řadič na MK60 podporuje celkem 16 obousměrných EP.

O přístup k registrům řadiče se na straně aplikačního programu stará obslužný software, který je často distribuován přímo výrobcem řadiče nebo daného MCU. Pro MK60 je k dispozici volně dostupný *Freescale USB Stack*, tj. sada knihoven, která obstarává řízení USB řadiče, předávání dat mezi řadičem a pamětí, implementuje standardní USB třídy a povinné popisovače, na základě kterých probíhá identifikace zařízení v systému USB hosta. Díky standardizaci popisovačů a tříd je možné na straně hosta používat pro různá zařízení jednotný ovladač. *Freescale USB Stack* podporuje následující třídy:

- Personal healthcare device class (PHDC)
- Human interface device (HID)
- Mass storage device (MSD)
- Communications device class (CDC)
- Audio class
- On-The-Go USB 2.0 Standard Supplement
- Vlastní uživatelské třídy

Zajímavými kandidáty pro přenos souborových dat jsou třídy MSD a CDC. Možné by bylo také vytvoření vlastní uživatelské třídy nebo přidání podpory pro standardní třídu Device Firmware Update (DFU), která slouží k aktualizaci software u USB zařízení. Největším úskalím v obou těchto případech by však bylo vytváření a integrace ovladačů pro různé operační systémy. Z tohoto důvodu se jeví jako výhodnější použiti stávajících standardních tříd.

Třída MSD

Tato třída je určena pro zařízení, která přenášejí soubory v jednom či obou směrech. Typickými představiteli jsou disketová jednotka, pevný disk, CD, DVD a různé typy vyjímatelných pamětí. USB zařízení s touto třídou se do systému začlení jako externí disk (Windows) nebo složka (Linux) a dále je s ním možné pracovat jako s jakýmkoliv jiným adresářem. Zařízení musí implementovat nějakou formu souborového systému (nejčastěji FAT). Třída MSD dále umožňuje USB hostovi zjistit, jaký typ protokolu je podporován. V případě externích disků a flash pamětí se používá sada příkazů SCSI (Small Computer Command Set). Výsledkem je možnost přesouvat, kopírovat a mazat soubory přímo v USB zařízení [6].

Z hlediska zavaděče je tato třída zajímavá, protože nahrání uživatelského software do paměti MCU není nic jiného, než přenos souboru z vývojového prostředí do paměti vestavěného systému. Použitím této třídy by se nahrání firmware omezilo na kopírování souboru do externího zařízení pomocí systémových prostředků. Firma Freescale již tento princip úspěšně vyzkoušela v jednom ze svých zavaděčů, jehož použitím na platformě Minerva se zabývá bakalářská práce Michala Dohnala [7], [8]. Použití tohoto principu v novém zavaděči je také možné, ale navzdory tomu, že se jedná o univerzální řešení, byla jeho implementace po dohodě s vedoucím práce ponechána jako budoucí možné rozšíření zavaděče.

Třída CDC

Tato třída sdružuje dva typy zařízení – telekomunikační zařízení (modemy, ISDN terminály, telefony) a síťová komunikační zařízení (ADSL modemy, kabelové modemy, Ethernetové adaptéry…). Primárním využitím je přenos aplikačně specifických protokolů v závislosti na zařízení a dnes se s ní můžeme často setkat také jako s náhradou pro asynchronní sériovou linku RS-232. Je to dáno tím, že zařízení podporující tuto USB třídu se v systému USB hosta zobrazí jako COM port, tedy stejně, jako

zařízení komunikující skrze fyzický RS-232 port. Na straně hosta lze proto pro komunikaci s oběma typy zařízení využít tentýž software.

Tento princip je výhodný i z pohledu nového zavaděče, protože díky němu je možné oběma zvažovanými rozhraními RS-232 i USB komunikovat stejným protokolem. V důsledku toho se zjednodušuje návrh aplikace na straně USB hosta i návrh zavaděče na straně MCU. Tato třída je tedy primárně použita v novém zavaděči.

3.4 Rozhraní Ethernet

Rozhraní Ethernet je na platformě Minerva složeno ze dvou komponent – kontrolérem pro přístup k médiu (MAC) integrovaným v MCU a řadičem síťového rozhraní (PHY) LAN8720 umístěným na desce plošných spojů samostatně. Obě komponenty jsou propojeny sběrnicí RMII, která používá 2 vodiče pro přenos paketových dat a je taktována na frekvenci 50 MHz.

Kontrolér MAC se stará o odesílání a příjem dat ze síťového rozhraní a kromě toho umožňuje akcelerovat zpracování paketů na úrovni síťových protokolů IP, TCP, UDP a ICMP. Hlavní výhodou je výpočet a ověření kontrolních součtů jednotlivých protokolů v paketu napříč protokolovým zásobníkem [1].

Podobně jako v případě USB řadiče i rozhraní MAC řadiče vystupuje v systému jako sada registrů mapovaných do adresového prostoru procesoru. Rozdíl je však v přístupu k řadiči síťového portu, jehož registry jsou dostupné prostřednictvím registrů MAC řadiče a rozhraní SMI. Pro oba řadiče jsou k dispozici od firmy Freescale potřebné ovladače pod licencí BSD. Tyto ovladače nicméně umožňují pouze odeslat či přijmout Ethernetový rámec a získat stav linky. Pro plnohodnotné fungování síťového rozhraní je potřeba dodat další knihovny, které se budou starat o komunikaci se službami běžícími v TCP/IP síti, jako je např. překlad adres ARP, DNS nebo služba DHCP a umožní aplikaci běžící v MCU komunikovat prostřednictvím síťových protokolů TCP, UDP a ICMP.

Pro tyto účely byla vybrána knihovna *LwIP*, která je velmi dobře konfigurovatelná a je svými nízkými paměťovými nároky navržená pro použití ve vestavěných zařízeních. Navíc je poměrně dobře optimalizovaná - například umožňuje práci s pakety bez nutnosti jejich kopírování mezi jednotlivými vrstvami a není problém ji používat i bez operačního systému. K dispozici je zdarma pod licencí BSD [9].

Použitím knihovny *LwIP* bude zavaděči umožněno komunikovat v paketové síti Ethernet. Na aplikační úrovni bude výhodné použít stejný protokol jako v případě rozhraní USB/RS-232, takže nebude narůstat objem kódu potřebný k realizaci všech funkcí zavaděče. Na transportní vrstvě se jeví jako nevýhodnější použít protokol TCP, který zaručí spolehlivé doručení dat v pořadí mezi aplikací na PC a platformou Minerva. Použití UDP by sice bylo také možné, ale vzhledem k tomu, že se předpokládá přenos většího objemu dat, než je velikost jednoho Ethernetového paketu (1500 B), tak

by bylo potřeba zajistit doručení dat v pořadí v rámci aplikačního protokolu, čímž by se návrh tohoto protokolu značně zkomplikoval.

3.5 Rozhraní RS-232

Toto rozhraní, tak jak jej definuje standard TIA-232-F, se na platformě Minerva nenachází, jelikož na plošném spoji není osazen ani 9pinový ani 25pinový D-Sub konektor, ani převodník z úrovně signálů CMOS/TTL na úroveň signálů RS-232 (výstupní napětí mezi +5 a +15V pro přenos log. 0 a -5V až -15V pro přenos log. 1) [10].

Sériové přenosy dat, které se na rozhraní RS-232 odehrávají, je však možné realizovat dvěma způsoby. První je sériový přenos z MK60 pomocí periferie UART5 do mikrokontroléru HCS08, kde dochází k zapouzdření dat do USB CDC třídy a k jejich přenosu do PC připojeného skrze USB rozhraní. Druhou možností je přenos z MK60 pomocí periferie UART4 do obvodu VNC2, který (se správným programem) také provede zapouzdření sériově přenášených dat pomocí USB CDC třídy a jejich zpřístupnění na straně USB hosta formou virtuálního COM portu. Z nabízených možností bylo po dohodě s vedoucím práce rozhodnuto, že využita bude sériová komunikace přes periferii UART5 a MCU HCS08. V dalším textu se tedy pod rozhraním RS-232 rozumí komunikace skrze tuto UART periferii.

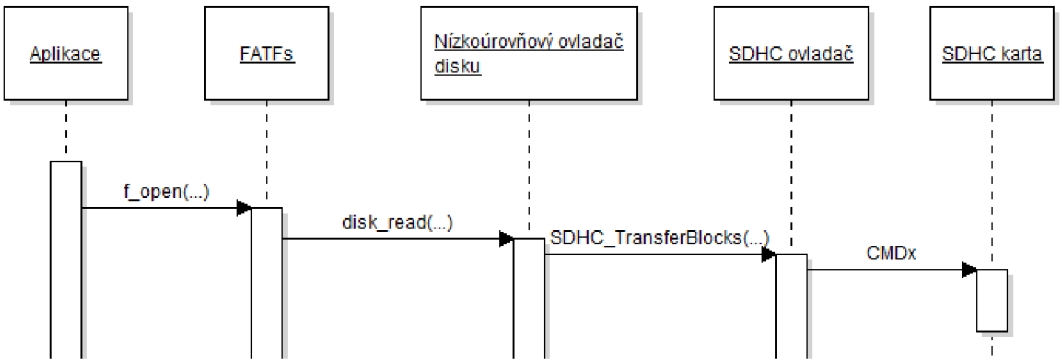
Vzhledem k tomu, že sériová linka se nachází na úrovni fyzické vrstvy přenosu dat a protokoly vyšších vrstev mohou být libovolné, bude potřeba zabezpečit aplikační protokol tak, aby vlivem případného rušení nemohlo dojít k chybě přenosu dat mezi mikrokontroléry HSC08 a MK60 (byť vzdálenost, ve které se komunikuje, je v řádu jednotek centimetrů).

3.6 Rozhraní SDHC

Toto rozhraní slouží k rozšíření paměťové kapacity platformy Minerva pomocí SD/SDHC karet. O komunikaci s paměťovou kartou se stará hostitelský řadič SDHC, který je jako periferie integrován do MK60. Řadič odpovídá definovaným SD standardům a umí komunikovat i s SD kartami s vysokou kapacitou, tzv. SDHC kartami. SD karty se zasouvají do konektoru na zadní straně platformy. Tento konektor je vybaven detekcí přítomnosti karty a detekcí mechanického povolení zápisu na kartu. Tyto informace jsou předávány na vstupně-výstupní piny MK60 pomocí signálů SDCARD_DET a SDCARD_WP. Tyto signály jsou pomocí odporů pull-up drženy na napěťové úrovni 3.3V, pokud není karta vložena. Signály jsou tedy aktivní při napětí 0V. Pomocí sestupné/vzestupné hrany na těchto signálech lze detekovat vložení/vyjmutí SD karty z konektoru a stav přepínače povolení zápisu.

Aby bylo možné z SD karty číst soubory, je třeba přistupovat k souborovému systému na ní umístěném. Na SD kartách se velmi často používají souborové systémy FAT, pro jejichž využití ve

vestavěných systémech existuje knihovna *FatFs*. Tato knihovna komunikuje prostřednictvím ovladače hostitelského řadiče SDHC s SD kartou, ze které čte bloky dat a následně je interpretuje jako části souborového systému FAT (tj. zaváděcí sektor, alokační tabulka souborů, kořenový adresář…). Ukázka použití knihovny *FatFs* je vidět formou sekvenčního diagramu na obrázku 3.3.

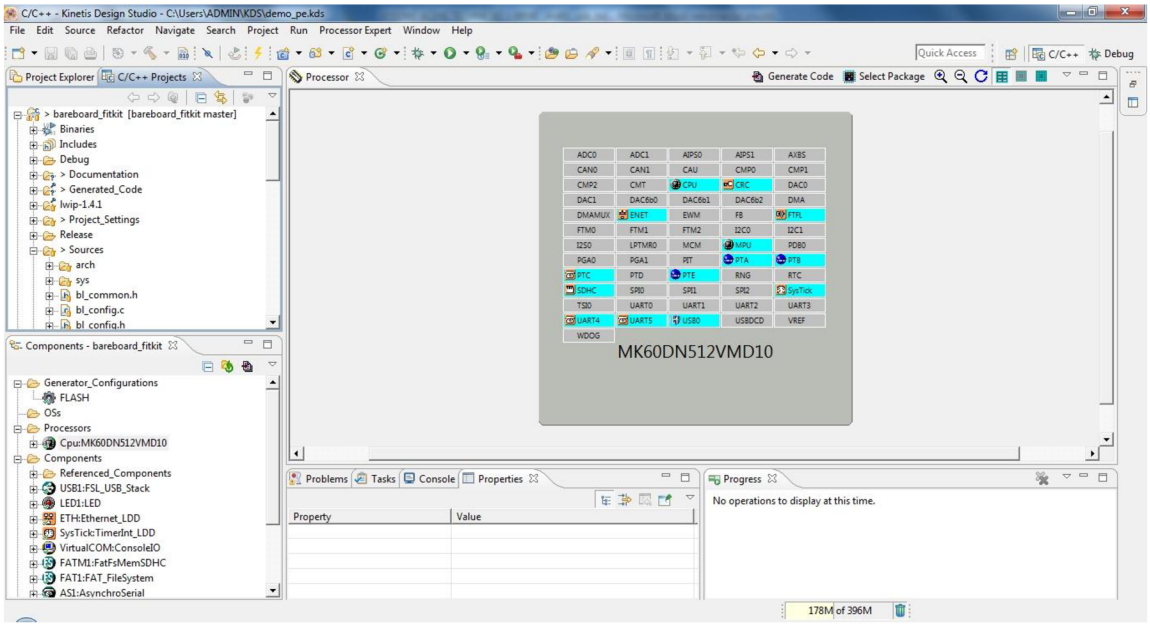


3.3 Začlenění knihovny FatFs v aplikaci.

Nový zavaděč by měl umět z SD karty s FAT souborovým systémem zkopírovat soubor s uživatelským programem do vnitřní paměti MK60 a poté jej spustit. K tomu bude potřeba do systému začlenit všechny části řetězce - knihovnu *FatFs*, nízkoúrovňový ovladač disku a ovladač SDHC kontroléru. Během zavádění uživatelského programu do MCU není potřeba provádět na SD kartu žádné zápisové operace, takže lze části knihovny *FatFs* provádějící zápis úplně odebrat a tím ušetřit cenný paměťový prostor.

4 Vývojové nástroje

Vyvíjet software pro mikrokontroléry Kinetis je možné v několika integrovaných vývojových prostředích, z nichž nejzajímavější je prostředí Kinetis Design Studio (KDS), aktuálně ve verzi 3.1.0, protože je dobře vybavené a je zcela zdarma. Toto prostředí je založeno na volně dostupné vývojové platformě Eclipse. Součástí je také sada nástrojů GNU tools for ARM® Embedded Processors, která obsahuje překladač GCC pro procesory s jádrem ARM a linkovací nástroj LD.



4.1 Kinetis Design Studio 3.1.0

Abstrakci nad hardware mikrokontrolérů Kinetis zajišťují dvě komponenty, které lze do vývojového studia přidat – Kinetis Software Development Kit (KSDK) a Processor Expert.

KSDK obsahuje tzv. HAL[1] vrstvu zastřešující hardwarové rozhraní MCU na nejnižší úrovni a ovladače pro jednotlivé periferie, dále protokolové zásobníky pro USB a Ethernet, operační systémy reálného času MQX a FreeRTOS a různé ukázkové aplikace. SDK je navrženo v souladu se standardem CMSIS pro přenositelnost zdrojových kódů napříč různými ARM platformami.

Processor Expert je doplněk vývojového prostředí, který usnadňuje rychlé prototypování aplikací pomocí konfigurovatelných šablon zdrojových kódů zvaných komponenty. Toto je jeden ze způsobů, jakým firma Freescale distribuuje ovladače z KSDK, ovšem pomocí tohoto nástroje lze do KDS přidávat i komponenty třetích stran. Komponenty mohou tvořit hierarchii, takže lze s malým úsilím velmi rychle získat základní kostru programu [11].

[1] Hardware Abstraction Layer

Pro ladění a krokování programu se využívá služeb GDB klienta, který zajišťuje zobrazení aktuálního stavu vykonávání programu v mikrokontroléru na straně vývojového prostředí. Tento klient komunikuje prostřednictvím síťového rozhraní s GDB serverem běžícím na lokálním/vzdáleném PC. Tento server pak dále komunikuje např. pomocí rozhraní USB s mikrokontrolérem provádějícím překlad příkazů dedikovaného protokolu na příkazy pro ladící hardware vestavěný v cílovém MCU. Přenos příkazů probíhá buď pomocí protokolu JTAG nebo SWD. V cílovém MCU jsou tyto příkazy zpracovány podpůrným ladícím hardware (např. ARM CoreSight Debug Access Port) a na základě nich jsou provedeny odpovídající akce. Těmi může být např. vyčtení hodnoty registru, spuštění programu apod. Odpověď na příkaz jde stejnou cestou zpět ke GDB klientovi.

Ladící nástroj OSBDM, který prostředí Kinetis Design Studio využívá při práci s platformou Minerva, byl vyvinut firmou P&E Micro a je volně dostupný včetně schématu zapojení a příslušného software. Bohužel v kombinaci s KDS je proces nahrávání a ladění programu na platformě Minerva poměrně pomalý navzdory různým nastavením parametrů komunikace (cca 30 sekund pro nahrání a spuštění programu o velikosti 64 KB).

K samotnému vývoji zavaděče mi však bylo prostředí KDS doporučeno vedoucím práce a i přes občasný menší komfort práce v něm (pomalá odezva) se jedná o velmi kvalitní vývojové prostředí s mnoha funkcemi. Největší výhodou je možnost snadné integrace existujících knihoven a funkcí formou komponent v modulu Processor Expert.

5 Návrh firmware

V této kapitole je na základě poznatků z předchozích kapitol proveden návrh a popsána implementace softwarové architektury nového zavaděče. Ta je z části inspirována stávajícími zavaděči představenými v kapitole 2.4.

5.1 Klíčové vlastnosti

Mezi klíčové vlastnosti nového zavaděče patří:

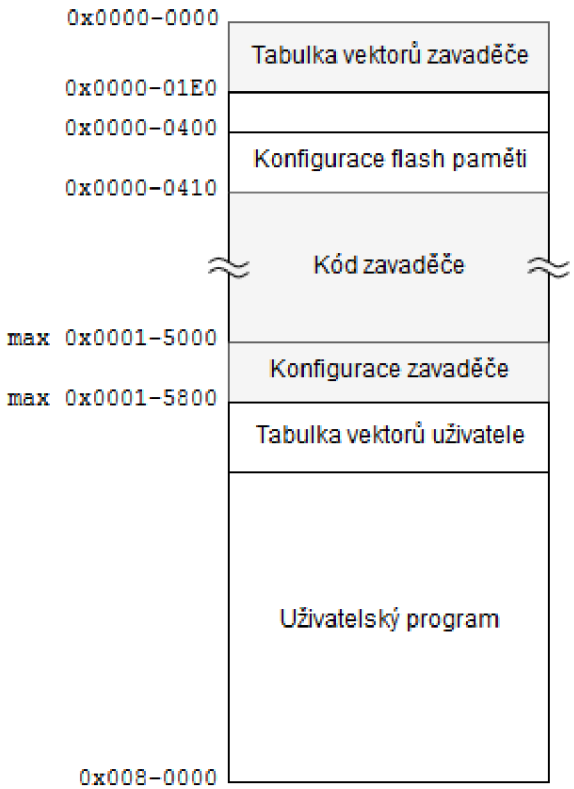
- Jednotný komunikační protokol na všech komunikačních rozhraních.
  - Jednotná vrstva pro komunikaci mezi jádrem zavaděče a periferiemi
- Podporovaná rozhraní:
  - Ethernet (TCP)
  - SDHC (FAT)
  - USB (CDC)
  - UART
- Detekce zavaděče na síti Ethernet
- Statická konfigurace při překladu
- Dynamická konfigurace z PC (Windows, Linux).
- Nahrávání a stahování uživatelského programu z PC (Windows, Linux)
- Implementace v jazyce C v prostředí KDS s podporou komponent modulu Processor Expert a podporou standardní knihovny funkcí jazyka C
- Velikost zavaděče do 96 KB

Podle rozdělení v kapitole 2.3 spadá nový zavaděč do kategorie samostatných zavaděčů. Jeho umístění v paměti MCU zobrazuje obrázek 5.1. Pro zavaděč je vyhrazena oblast na začátku flash paměti, která může být různé dlouhá v závislosti na statické konfiguraci zavaděče. Jejím účelem je minimalizace paměťových nároků a odstranění aktuálně nevyužívaných částí. Staticky je možné konfigurovat následující parametry:

- Dostupná komunikační rozhraní (Ethernet, SDHC, USB, UART)
- Podpora DHCP pro rozhraní Ethernet
- Podpora programování FPGA[2]

Za kódem zavaděče se nachází 2 KB sektor dynamické konfigurace, který slouží k uložení atributů zavaděče (nastavení IP adresy, volba spouštění) a parametrů nahraného programu (CRC,

[2] Jedná se o rozšíření zavaděče nad rámec zadání, které bylo převzato z diplomové práce Petra Buchty [13].



5.1 Umístění zavaděče a uživatelského programu v paměti mikrokontroléru

délka). Ke změně parametrů je nutné využít některou z připravených utilit pro komunikaci se zavaděčem běžících na PC – není podporována jejich změna prostřednictvím SD karty.

Za konfigurací zavaděče začíná oblast uživatelského programu, která je shora omezena velikostní vnitřní flash paměti. Na začátku této oblasti se musí nacházet tabulka vektorů přerušení uživatele, kterou zavaděč nastaví jako aktivní při přechodu do uživatelského programu. Tabulka vektorů přerušení zároveň obsahuje adresu první instrukce uživatelského programu, na kterou zavaděč skočí během přechodu do uživatelského programu.

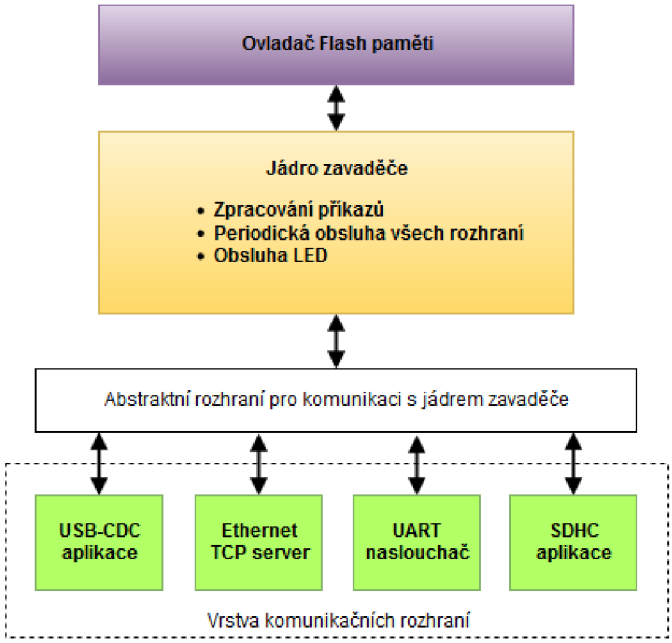
Uživatelský program může pro své potřeby používat nevyužitou část flash paměti alokovanou uživatelskému programu při dodržení výše popsaných omezení. Takto uložená data však nejsou zavaděčem zachována v případě, že dojde k nahrání nového programu či nové verze stávajícího programu. Uživatelský program taktéž nesmí modifikovat obsah flash paměti alokovaný zavaděčem. V případě potřeby lepšího zabezpečení obsahu flash paměti by bylo možné např. uzamknout oblast zavaděče proti zápisu, čímž by se efektivně zamezilo modifikaci této paměti ze strany programu. Tato možnost je ponechána jako možné rozšíření zavaděče.

Na všech komunikačních rozhraních zavaděče je podporován jednotný binární paketový komunikační protokol, aby se dále minimalizovaly paměťové nároky, zvětšilo se znovupoužití kódu a zvýšila se univerzálnost zavaděče. Návrh počítá se snadným rozšířením o případná další podporovaná rozhraní. Více informací o komunikačním protokolu se nacházejí v kapitole 5.4.

Pro lepší detekci zavaděče byl implementován jednoduchý UDP protokol pro zjišťování běžících zavaděčů na lokální síti Ethernet. Informace k tomuto protokolu se nacházejí v kapitole 5.5.2. Informace o maximální velikosti uživatelského programu s konkrétními překladovými konfiguracemi zavaděče se nachází v kapitole 5.7.

5.2 Struktura zavaděče

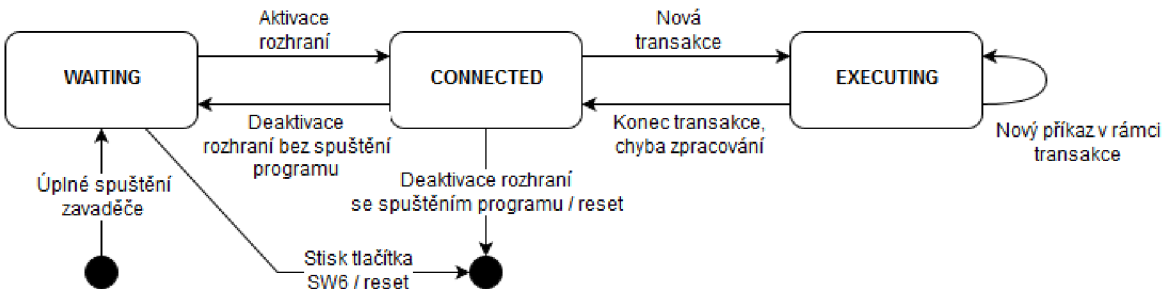
Vzhledem k použití jazyka C je nový zavaděč naprogramován jako modulární aplikace. Jednotlivé moduly mají přesně definované funkce. Schéma zavaděče se nachází na obrázku 5.2.



5.2 Blokové schéma nového zavaděče

Modul jádra zavaděče

Tento modul je nejdůležitějším v celém zavaděči. Obsahuje definice struktur všech podporovaných periferií, definice podporovaných příkazů a procedury pro jejich obsluhu. Pokud je požadováno úplné spuštění zavaděče, mění se jeho stav podle následujícího stavového diagramu:



5.3 Stavový diagram jádra zavaděče

Jednotlivé stavy mají následující význam:

- **WAITING** – zavaděč čeká neomezeně dlouho na navázání komunikace na některém z dostupných rozhraní. Stiskem tlačítka v tomto stavu je možné vyvolat reset, po kterém proběhne pokus o spuštění stávajícího programu.
- **CONNECTED** – zavaděč čeká na zahájení transakce (zaslání příkazu) na připojeném rozhraní. Pokud dojde k odpojení všech rozhraní, vrací se zavaděč do stavu WAITING nebo provede reset MCU, po kterém se pokusí spustit uživatelský program. Volba záleží jednak na nastavení parametrů zavaděče a dále také na tom, zda byl během připojení zaslán příkaz pro spuštění uživatelského programu (viz dále).
- **EXECUTING** – probíhá provádění transakce na jednom z připojených rozhraní. Lze provádět pouze jednu transakci zároveň. V tomto stavu zavaděč zůstává, dokud nedojde k provedení všech příkazů transakce nebo nedojde k chybě při provádění transakce. Podrobnější popis transakce se nachází v kapitole 5.4.

Tento modul také provádí inicializaci a obsluhu všech komunikačních rozhraní a definuje funkci, která signalizuje běh zavaděče pomocí diody MCU_LED0. Význam stavu diody je následující:

- LED kontinuálně mění intenzitu svitu - zavaděč se nachází ve stavu WAITING nebo CONNECTED.
- LED svítí konstantní intenzitou – zavaděč se nachází ve stavu EXECUTING.

Modul ovladače flash paměti

Úlohou tohoto modulu je spojování (deserializace) jednotlivých částí přenášeného souboru a jejich zápis do vnitřní flash paměti MCU (transakce zápisu do flash), resp. načítání těchto částí (serializace) pro přenos do aplikace na straně hosta (transakce čtení z flash). Konec souboru je signalizován přenosem zprávy s datovou částí o velikosti dat 0 bajtů.

Dalším úkolem, který tento modul provádí, je mazání flash paměti před zápisem, které je podmíněno principem funkce flash paměti. Důležitými faktory jsou proto velikost sektoru (2 KB), který určuje nemenší úsek paměti, který lze vymazat, a velikost stránky (4 B), což je nejmenší úsek paměti, který lze zapsat. Přestože lze u MK60 zapisovat do flash paměti i jednotlivá dvojslova, je z hlediska rychlosti zápisu vždy lepší zapisovat co nejvíce dat najednou. Pro uložení uživatelského programu je tedy vyhrazen celočíselný násobek počtu sektorů a část souboru přenášená v jednom datovém paketu vždy odpovídá velikosti sektoru – tj. 2 KB. Výjimkou je pouze poslední datový paket, který může být kratší.

Abstraktní rozhraní pro komunikaci s jádrem zavaděče

Toto rozhraní slouží ke sjednocení formátu příkazů a parametrů předávaných jádru zavaděče z různých komunikačních rozhraní. Každé komunikační rozhraní, které chce s jádrem zavaděčem

komunikovat, musí toto softwarové rozhraní implementovat. Rozhraní je definováno následujícími typy ukazatelů na funkce, resp. prototypy funkcí vyhovující typům těchto ukazatelů:

```
/* Funkce zavolaná při změně stavu rozhraní (připojeno/odpojeno) */
typedef void (* app_link_cb)(bool state, void * arg);

/* Funkce zavolaná při přijetí příkazu na rozhraní */
typedef void (* app_recv_cb)(const char * data, size_t length, void *
arg);

/* Funkce pro odeslání dat na rozhraní */
typedef int  (* app_send_fn)(const char * data, size_t length);

/* Funkce periodické obsluhy rozhraní */
typedef void (* app_poll_fn)(void);

/* Inicializační funkce */
typedef int  (* app_init_fn)(app_recv_cb app_cb, app_link_cb link_cb, void
* arg);

/* Deinicializační funkce */
typedef void (* app_deinit_fn)();
```

V rozhraní jsou rozlišeny prototypy klasických funkcí (s příponou _fn) a funkcí typu *callback* (s příponou _cb). Rozdíl mezi nimi je ten, že klasické funkce jsou implementovány v nižších softwarových vrstvách (ovladačích daných rozhraní) a jsou volány z jádra zavaděče, zatímco *callback* funkce jsou volány z ovladače rozhraní a jsou implementovány ve vyšších softwarových vrstvách (v jádře zavaděče). Tento princip předávání dat jasně vymezuje hranice mezi jednotlivými softwarovými vrstvami a zachovává pravidlo viditelnosti nižších vrstev z vyšších a nikdy ne naopak. Díky tomu se zlepšuje modularita celého systému – moduly na nižších úrovních mohou být snadno znovupoužity v jiných aplikacích.

Komunikační rozhraní je inicializováno pomocí inicializační funkce, která má jako jeden z argumentů *arg*. Pomocí tohoto argumentu rozlišuje zavaděč mezi jednotlivými komunikačními rozhraními a ta jej musí používat u všech volání výše uvedených funkcí. Dále se při inicializaci nastavují ukazatele na *callback* funkce zavaděče.

Vrstva komunikačních rozhraní

Komunikační rozhraní jsou implementovány jako izolované aplikace běžící v rámci zavaděče. Každá aplikace se stará o komunikaci pomocí specifické periferie (UART, USB…). Pro zjednodušení návrhu zavaděče používají všechny aplikace jednotnou strukturu *link_ctx* pro uložení stavu rozhraní, ukazatelů na funkce pro komunikaci s jádrem zavaděče, buffer pro data, odesílací frontu, časové známky atp. Tato struktura je částečně znázorněna na obrázku 5.4.

```
typedef struct link_ctx
{
      /* Funkce pro předání přijatých dat vyšší vrstvě */
      app_recv_cb app_data_cb;
      /* Funkce pro indikaci změny stavu rozhraní vyšší vrstvě */
      app_link_cb app_linkstate_cb;
      /* Parametr identifikující rozhraní */
      void * app_cb_arg;
      /* Přijímací buffer pro celou zprávu */
      char rx_buffer[sizeof(link_header_t) + MESSAGE_MAX_LEN +
                                          FRAME_CHECKSUM_LEN];
      /* Aktuální délka přijímaných dat */
      uint32_t rx_length;
      /* Časová značka přijímaných dat */
      uint32_t rx_timestamp;
      ...
      /* Fronta odesílaných zpráv */
      SIMPLEQ_HEAD(tx_squeue, tx_sqentry) tx_sqhead;
} link_ctx_t;
```

5.4 Struktura popisující komunikační rozhraní

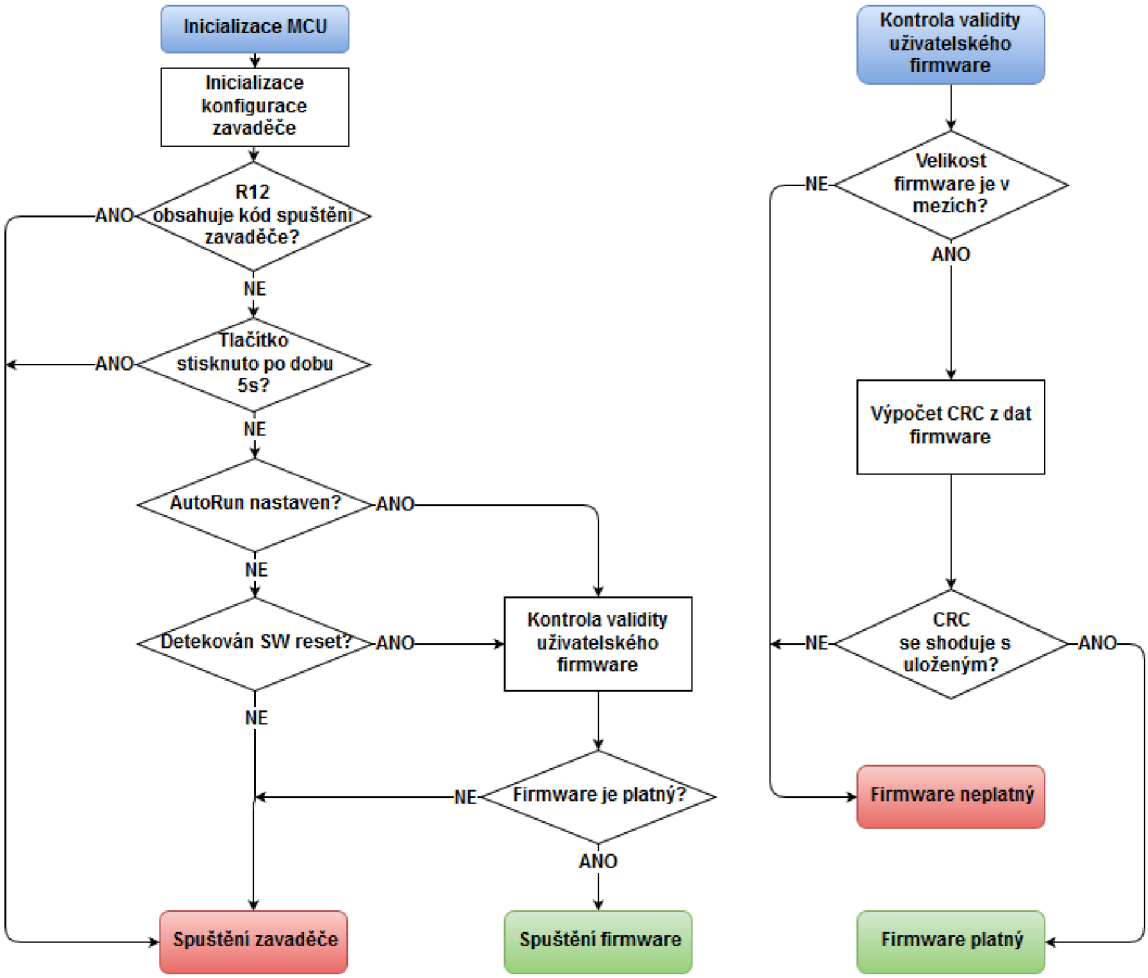
5.3 Spouštění zavaděče

Rozhodnutí o spuštění zavaděče záleží na následujících faktorech seřazených podle pořadí jejich vyhodnocení:

1) Spuštění zavaděče pomocí rozhraní OSBDM
2) Stisk tlačítka během resetu či připojení napájení
3) Parametr AutoRun v konfiguraci.
4) Typ resetu
5) Platnost nahraného firmware

Vývojový diagram 5.5 zobrazuje posloupnost vyhodnocení těchto faktorů a s nimi spojených akcí. Vyhodnocení probíhá těsně po inicializaci mikrokontroléru. Výsledkem je buď rozhodnutí o spuštění nahraného uživatelského programu, nebo úplné spuštění zavaděče s inicializací všech rozhraní a příprava na nahrání nového programu.

Prvním rozhodovacím krokem je požadavek na spuštění zavaděče pomocí ladícího rozhraní OSBDM. Tento požadavek má nejvyšší prioritu, protože může být vyvolán jenom uživatelem a to v jakémkoliv stavu MCU, tedy i v případě, že běží uživatelský program. Je využito toho, že OSBDM má asynchronní přístup k mikrokontroléru a kdykoliv může provést jeho reset z vnějšku, dále může pozastavit jeho běh a také může libovolně zapisovat do jeho registrů či paměti. OSBDM tedy provede reset MCU do ladícího režimu, ve kterém má nad ním absolutní kontrolu a následně zapíše do registru R12 speciální 32bitovou konstantu, čímž signalizuje úvodní rutině zavaděče, že má pokračovat ve spouštění zavaděče. Podle specifikace ARM nemá registr R12 definovaný obsah po resetu, takže lze obecně předpokládat libovolnou hodnotu [12]. Pravděpodobnost, že se po resetu bude v registru nacházet právě hodnota spouštějící zavaděč, je téměř nulová (2,3e-10). Po provedení těchto kroků



5.5 Vývojový diagram spouštění zavaděče a vyhodnocení platnosti nahraného firmware

rozhraní OSBDM spustí vykonávání programu zrušením ladícího režimu v odpovídajících registrech MCU, čímž spustí zavaděč. Tento přístup má však dvě zásadní nevýhody – funguje pouze tehdy, pokud je rozhraní OSBDM fyzicky připojeno k lokálnímu počítači a zároveň k němu není připojena žádná jiná aplikace (P&E GDB Server…). Zároveň se však jedná o jedinou možnost jak spolehlivě vyvolat zavaděč v jakémkoliv stavu MCU.

Druhým možným způsobem spuštění zavaděče je stisk tlačítka během připojení napájení. K tomuto účelu bylo na platformě Minerva vyhrazeno tlačítko SW6. Tlačítko je nutné stisknout před připojením napájení (před připojením USB kabelu) a držet nejméně po dobu 5 sekund po připojení napájení. Pokud úvodní rutina detekuje stisknuté tlačítko, opět přeskočí další detekční fáze a spustí zavaděč. Podobný způsob se dnes používá pro spuštění zavaděče u mobilních telefonů a jeho výhodou a zároveň nevýhodou je jeho jednoduchost. Pokud je spouštění zavaděče ojedinělou událostí, pak se jedná o velmi efektivní řešení, které funguje bez přídavného software. Probíhají-li však aktualizace poměrně často, například během vývoje uživatelského programu, pak je tento postup hodně nepraktický, protože vyžaduje ruční manipulaci s vývojovou deskou.

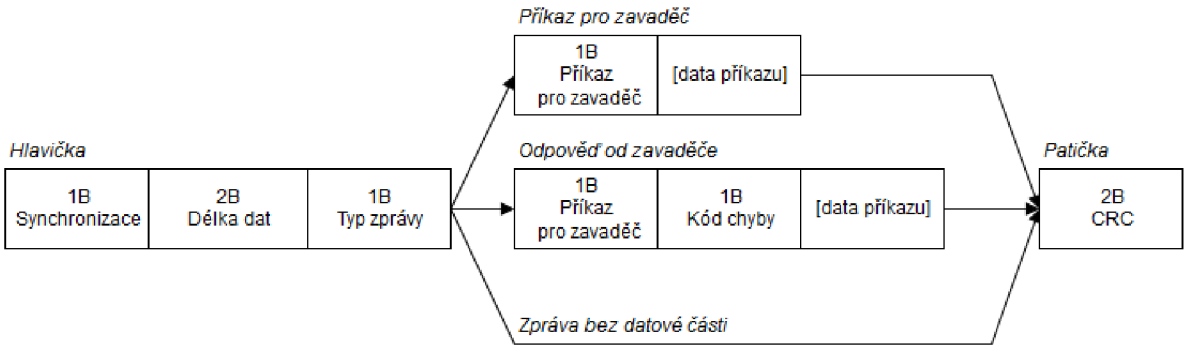
Třetí možností je využití parametru AutoRun, který nabývá hodnot 0 či 1 a je uložen v dynamické konfiguraci zavaděče. Jeho nastavení ovlivňuje automatické spuštění zavaděče po resetu. Je-li parametr nastaven na hodnotu 1, pak zavaděč zkouší detekovat nahranou uživatelskou aplikaci a je-li přítomna, tak ji automaticky spustí. V opačném případě proběhne spuštění zavaděče i přesto, že v MCU může být nahrán platný uživatelský program. Smyslem tohoto parametru je umožnění volitelného spuštění zavaděče či uživatelského programu, například během jeho vývoje.

Čtvrtou možností, jak vyvolat zavaděč, je pomocí různého typu resetu. Při softwarovém resetu vyvolaném z uživatelské aplikace zavaděč opět spouští uživatelskou aplikaci. Při resetu vzniklém při výpadku napájení se vždy spouští zavaděč, není-li definováno jinak pomocí parametru AutoRun.

Pokud má dojít ke spuštění uživatelského programu, ale validitu jeho uložení v interní flash paměti se nepodaří ověřit, pak se také spustí zavaděč. Během ověření je kontrolováno, zda je velikost nahraného programu nenulová a zároveň menší než maximální možná velikost, a zda se shoduje jeho CRC s CRC zaznamenaným do dynamické konfigurace zavaděče během jeho nahrávání. Pokud jsou podmínky splněny, je program prohlášen za platný a může být spuštěn.

5.4 Komunikační protokol

Protokol pro přenos příkazů a odpovědí je velmi jednoduchý. Každá zpráva se sestává z hlavičky, volitelné datové části a patičky. Vícebajtová pole protokolu jsou přenášena podle schématu Little-Endian, není-li specifikováno jinak. Formát zprávy je zobrazen na následujícím obrázku:



5.6 Formát zprávy protokolu zavaděče

Hlavička

Hlavička se sestává ze 4 bajtů, z nichž první je synchronizační bajt s hodnotou 0x55, následují 2 bajty určující délku pouze datové části zprávy a 1 bajt určující typ zprávy. Možné hodnoty pole „Typ zprávy“ jsou rozepsány v příloze C.1.

Za zpracování hlavičky je vždy odpovědný vysokoúrovňový ovladač daného rozhraní. Protokol je tedy připraven v tomto ohledu i na možnost, že každé rozhraní bude používat jiný formát hlavičky. Nakonec se však z praktických důvodů ukázalo jako výhodnější použít u rozhraní USB, RS-232 a Ethernet formát hlavičky totožný. U rozhraní SDHC není potřeba hlavičku využívat vůbec, protože SDHC aplikace přímo komunikuje s jádrem zavaděče pouze pomocí interního rozhraní.

Datová část

V datové části je přenášen příkaz pro zavaděč, který je určen hodnotou prvního bajtu, za nímž následuje libovolně dlouhá sekvence bajtů tvořící parametr příkazu. Parametr je volitelný a jeho formát je závislý na konkrétním příkazu.

Druhou možností je přenos odpovědi od zavaděče pro dříve zaslaný příkaz. V takovém případě následuje za kódem příkazu ještě chybový kód určující výsledek provádění příkazu. Chybový kód 0 znamená, že nedošlo k žádné chybě a příkaz byl úspěšně proveden. Popisy všech chybových kódů mohou být nalezeny v souboru *PE_Error.h*, který je součástí projektu zavaděče. Popis podporovaných příkazů se nachází v příloze C.2.

Patička

Za datovou částí zprávy následuje vždy patička, která je složena ze dvou bajtů kontrolního součtu. Ten je vypočítán ze všech bajtů zprávy pomocí algoritmu CRC16-CCITT s počáteční hodnotou 0xFFFF a slouží k zajištění integrity přenášených dat. Pokud zaslaný kontrolní součet neodpovídá kontrolnímu součtu z přijatých dat, je zpráva ignorována a aplikace na PC musí toto detekovat pomocí uplynutí nastavené doby (časovače).

Transakce

Ve stávající verzi protokolu se transakce sestává z jednoho nebo více opakování téhož příkazu, který zasílá aplikace na PC či SDHC aplikace v zavaděči směrem k jádru zavaděče nebo opačně. Účelem transakce je nepřerušované zpracování série příkazů, které je potřeba provést v případě přenosu souboru s programem. Příkazy v rámci transakce musí být do zavaděče zasílány sekvenčně. Vždy před zasláním dalšího příkazu musí odesílající aplikace přijmout chybový kód předchozího příkazu. Transakce je ukončena v momentě, kdy je o tom rozhodnuto během jejího zpracování. Zavaděč rozlišuje mezi 4 typy ukončení transakce:

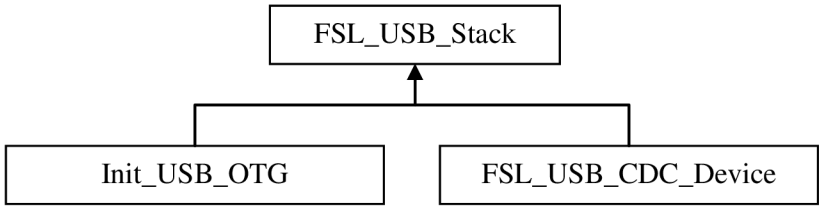
1) Dojde k úspěšnému ukončení transakce
2) Dojde k chybě během provádění transakce
3) Dojde k uživatelskému přerušení transakce příkazem ABORT
4) Dojde k odpojení komunikačního rozhraní

Pokud dojde k neúspěšnému ukončení transakce zápisu uživatelského programu do MK60, pak je uživatelská část vnitřní flash paměti vymazána z bezpečnostních důvodů.

5.5 Specifika komunikačních rozhraní

5.5.1 Rozhraní USB

Rozhraní USB je v zavaděči nakonfigurováno pomocí několika komponent modulu Processor Expert v prostředí KDS, které jsou hierarchicky uspořádány podle obrázku 5.7.



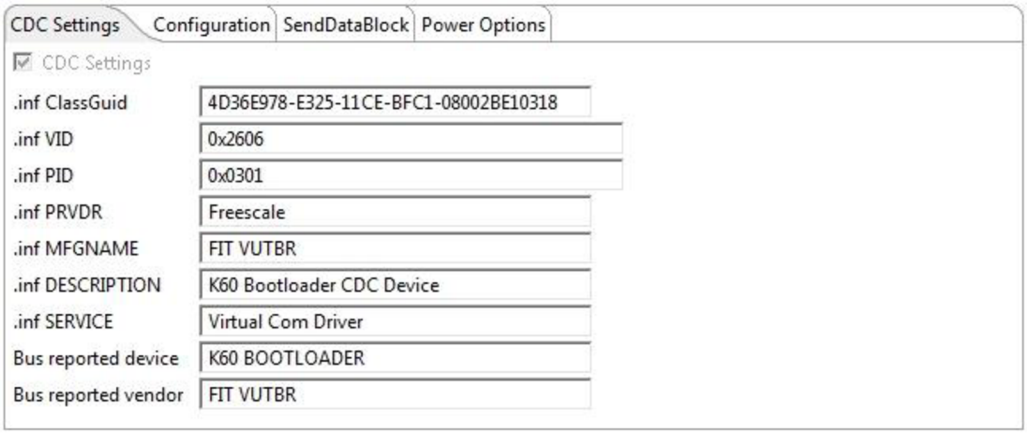
5.7 Hierarchie USB komponent modulu Processor Expert

Komponenta *FSL_USB_Stack* slouží k volbě požadované USB třídy. Tou je třída CDC zařízení, reprezentovaná komponentou *FSL_USB_CDC_Device*. Druhá podřazená komponenta je *Init_USB_OTG*, která reprezentuje USB řadič. Nadřazená komponenta *FSL_USB_Stack* tedy slouží k propojení kódu obou komponent a správnému nastavení všech USB deskriptorů a koncových bodů (end-point).

Součástí konfigurace komponenty *Init_USB_OTG* je nastavení zdroje hodinového signálu pro USB periferii tak, aby výsledný takt byl 48 MHz. Zdroj hodinového signálu musí být z důvodu přesnosti odvozen od výstupu obvodu frekvenčního (FLL) nebo fázového závěsu (PLL), který zároveň slouží k taktování jádra MCU a dalších periferií. Bylo tedy nutné nalézt takové nastavení hodinových rozvodů, aby mohlo jádro MCU běžet na nejvyšší možné frekvenci a zároveň bylo dodrženo omezení kladené na takt USB periferie.

Řešením je použití dvou různých hodinových konfigurací. První konfigurace odpovídá výchozímu nastavení MCU, při kterém je jádro taktováno z vnitřního IRC oscilátoru přibližně na 21 MHz. Tato konfigurace je aktivní po resetu mikrokontroléru a slouží k provedení základní inicializace a rozhodnutí o dalším běhu zavaděče nebo o spuštění uživatelského programu, který tak může při další konfiguraci hodin spoléhat na jejich výchozí nastavení. Pokud je rozhodnuto o běhu zavaděče, proběhne přepnutí na alternativní hodinovou konfiguraci využívající PLL. V tomto nastavení běží jádro MK60 na 96 MHz, což umožňuje dělením dvěma získat požadovaný takt pro USB periferii.

V komponentně *FSL_USB_CDC_Device* se nachází ještě nastavení související především se samotnou třídou USB CDC. Mezi nejdůležitějšími parametry patří nastavení identifikátorů zařízení (ProductID, PID), dodavatele (VendorID, VID) a dalších textových řetězců identifikujících USB zařízení na straně hosta. Jelikož získání VID identifikátoru je poměrně finančně náročné ($5000 dle webových stránek www.usb.org), byl proto použit pro účely testování náhodně zvolený a dosud nepřiřazený VID identifikátor 0x2606.



5.8 Konfigurace třídy USB CDC v zavaděči

Na základě těchto informací potom komponenta *USB_FSL_CDC_Device* vygeneruje ovladač pro prostředí Windows (soubor s příponou .inf), který bude požadován při prvním připojení zavaděče s povolenou USB periferií k systému.

V prostředí Linux je již potřebný ovladač součástí systému. Výpisem ze systémového logu (např. příkazem *dmesg*) stačí po připojení pouze ověřit, že se zavaděč správně identifikoval v systému. Následující příklad je z OS Ubuntu 14.04:

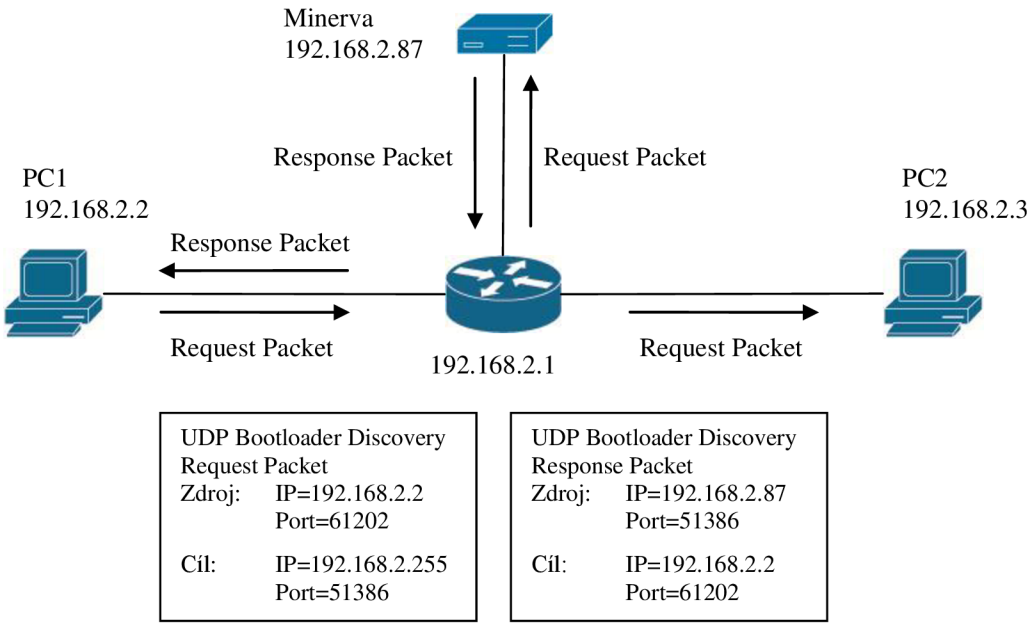
```
[  109.210145] usb 2-2: new full-speed USB device number 3 using ohci-pci
[  109.689739] usb 2-2: New USB device found, idVendor=2606, idProduct=0301
[  109.689744] usb 2-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[  109.689746] usb 2-2: Product: K60 BOOTLOADER
[  109.689748] usb 2-2: Manufacturer: FIT VUTBR
[  109.722282] cdc_acm 2-2:1.0: ttyACM0: USB ACM device
[  109.730101] usbcore: registered new interface driver cdc_acm
[  109.730105] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters
```

5.9 Výpis ze systémového logu po připojení zavaděče do Linuxového systému

5.5.2 Rozhraní Ethernet

Na rozhraní Ethernet je podporováno nastavení buď statické IP adresy nebo získávání IP adresy z DHCP serveru. Volba je předmětem dynamické konfigurace zavaděče. Mezi aplikací na straně PC a zavaděčem se pro konfiguraci a přenos souborů vytváří TCP spojení na portu 51387. Zavaděč

funguje v roli serveru, který umožňuje otevřít právě jedno TCP spojení klientské aplikaci. Pro udržování TCP spojení je povolen mechanizmus TCP keep-alive, který periodicky v 10sekundových intervalech zjišťuje průchodnost linky mezi komunikujícími stranami pomocí speciálních TCP keep-alive paketů.



5.10 Příklad použití detekčního protokolu, při vyhledávání připojených zavaděčů z PC

Pro snadnou detekci zavaděče na lokální síti je volitelně součástí zavaděče jednoduchý UDP detekční protokol, jehož princip je znázorněn na obrázku 5.10. Protokol používá pouze dva typy paketů – paket pro požadavek (Request Packet) a paket pro odpověď (Response Packet). Zavaděč naslouchá na příchozí všesměrové UDP pakety s požadavkem na portu 51386. Pokud se v datové části příchozího paketu nachází řetězec

„ << *K60BOOTLOADER_DISCOVERY_REQUEST* >> “,

pak zašle zavaděč na zdrojovou IP adresu tohoto paketu nový UDP paket s odpovědí ve formátu

„ << *K60BOOTLOADER_DISCOVERY_RESPONSE*>> “,

čímž sdělí vyhledávači svoji přítomnost pod specifickou IP adresou.

Vyhledávání je vzhledem k použití všesměrových paketů omezeno pouze na lokální síťový segment, ale to nepředstavuje závažnější problém vzhledem k předpokládanému nasazení zavaděče v převážně kancelářském či domácím prostředí. Standardizovanou alternativou tohoto přístupu by mohlo být použití protokolu Simple Service Discovery Protocol (SSDP) pro vyhledávání síťových zařízení a jejich služeb. Jeho možnosti však široce přesahují požadavky na pouhé zjištění IP adresy zařízení s konkrétní službou a jeho implementace do zavaděče by nebyla výhodná především z prostorového hlediska.

5.5.3 Zavádění programu pomocí SD karty

Zavádění programu pomocí SD karty probíhá podobně jako u jiných rozhraní, avšak s tím rozdílem, že aplikace, která zasílá příkazy zavaděči, je přímo jeho součástí (viz obrázek 5.2). Tato aplikace provádí inicializaci SDHC rozhraní, detekci připojení paměťové karty, inicializaci paměťové karty, načítání částí datového souboru s programem do příkazů komunikačního protokolu a jejich zasílání jádru zavaděče.

Aby bylo zavádění z pohledu uživatele co nejjednodušší, není potřeba při běžícím zavaděči dělat nic jiného než vložit paměťovou kartu do slotu a počkat určitou dobu, než zavaděč nakopíruje uživatelský firmware (soubor MINERVA_FW.BIN nacházející se v kořenovém adresáři) do vnitřní flash paměti. Poté buď dojde k automatickému spuštění programu (pokud byl parametr AutoRun dříve nastaven), nebo je možné jej spustit pomocí tlačítka SW6.

Paměťová karta tak může ve slotu zůstat i po spuštění uživatelského firmware a ten ji může dále využívat. Z bezpečnostních důvodů není během zavádění programu z paměťové karty možné připojení k zavaděči prostřednictvím jiných komunikačních rozhraní.

5.6 Ukončení činnosti zavaděče

Ukončení běhu zavaděče je nutné explicitně vyvolat uživatelem některou z následujících akcí:

1) Nastavením parametru AutoRun a zavřením komunikačního kanálu
2) Zasláním příkazu RUN a zavřením komunikačního kanálu
3) Stiskem tlačítka ve stavu WAITING

Zavření komunikačního kanálu je specifické podle použitého rozhraní. Na rozhraní Ethernet je za ukončení komunikace považováno ukončení TCP spojení se zavaděčem, ať už se jedná o korektní ukončení spojení nebo ukončení z důvodu chyby. V případě sériových rozhraní USB a RS-232 se komunikační kanál zavírá zasláním zprávy typu 0x00 – Ukončení komunikace. U rozhraní SDHC vnitřní komunikační kanál automaticky zavírá SDHC aplikace v zavaděči po dokončení programování uživatelské aplikace nebo při chybě.

Po zavření všech komunikačních kanálů zavaděč rozhoduje o spuštění nahrané aplikace. Pokud je parametr AutoRun nastaven, provede v tuto chvíli zavaděč kontrolu validity nahraného programu (viz obrázek 5.5) a pokud je platný, pak provede softwarový reset MCU a tím umožní jeho spuštění.

V případě spuštění programu pomocí příkazu RUN se v okamžiku přijetí příkazu pouze vyhodnotí platnost programu a zaznamená se požadavek na jeho spuštění po odpojení komunikačního rozhraní. Pokud je program neplatný, pak se to klientská aplikace dozví pomocí návratového kódu zaslaného v odpovědi příkazu a může na to reagovat např. zobrazením varovného hlášení uživateli.

Poslední možností spuštění nahraného firmware, je pomocí stisku tlačítka v okamžiku, kdy k zavaděči není připojena žádná klientská aplikace. Na platformě Minerva je k tomuto účelu použito tlačítko SW6, tedy stejné tlačítko jako pro spuštění zavaděče. I v tomto případě probíhá spuštění aplikace přes reset mikrokontroléru.

5.7 Metriky zavaděče

Tabulka shrnuje dosažené velikosti binárního kódu při kompilaci s různým nastavením statické konfigurace zavaděče. Kompilace byla prováděna se zapnutými optimalizacemi na velikost kódu (-Os). Velikost textové sekce zahrnuje také 2 KB stránku s konfigurací a její potřebné zarovnání na hranici sektoru flash paměti. Celková velikost binárního obrazu zavaděče je dána součtem velikosti sekcí Text a Data.

Konfigurace	Podporovaná rozhraní				Velikost zavaděče				Počáteční adresa uživatelského programu	Maximální velikost uživatelského programu
	Ethernet	USB	RS-232	SDHC	Text	Data	Bss	Celkem		
0	×	×	×	×	16244	4048	10688	19 KB	-	-
1	✓	×	×	×	70380	3266	42416	71 KB	0x00012000	440 KB
2	✓[3]	×	×	×	64680	2608	42224	65 KB	0x00010800	446 KB
3	×	✓	×	×	27016	3560	16388	29 KB	0x00007800	482 KB
4	×	×	✓	×	24148	4356	12788	27 KB	0x00007000	484 KB
5	×	×	×	✓	30516	4132	15200	33 KB	0x00008800	478 KB
6	✓	✓	✓	✓	81920	4068	54788	83 KB	0x00015000	428 KB

5-1 Tabulka překladových konfigurací zavaděče

Tabulka překladových konfigurací ukazuje, že se podařilo naplnit předpoklad maximální velikosti zavaděče do 96 KB. Maximální velikost uživatelského programu s uvedenými překladovými konfiguracemi se pak pohybuje v rozmezí 428-484 KB, což představuje zhruba 83-94 % dostupné flash paměti na mikrokontroléru MK60.

Z tabulky lze také odvodit, že paměťově nejnáročnější komponentou zavaděče je TCP/IP protokolový zásobník *LwIP*. Velikost paměti, kterou tento zásobník zabírá, se liší podle toho, jak dobře se podaří kompilátoru kód zásobníku optimalizovat s ohledem na další moduly. Významný podíl na tom má také zapnutá podpora DHCP (rozdíl 6 KB mezi konfiguracemi 2 a 3).

[3] Konfigurace bez DHCP klienta

Dále si lze všimnout, že součet celkové velikosti zavaděče a maximální velikosti uživatelského firmware je 511 KB místo očekávaných 512 KB. Chybějící 1 KB prostoru není započítán od celkové velikosti zavaděče – jedná se o první kilobajt flash paměti vyhrazený pro tabulku vektorů zavaděče.

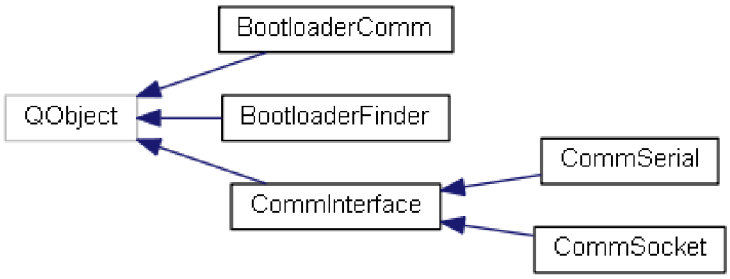
Z různých překladových konfigurací také vyplývá, že při přechodech mezi nimi se mění jednak počáteční adresa uživatelského programu ve flash paměti, a také adresa konfiguračního sektoru zavaděče. Při změně zavaděče je tedy potřeba provést jeho rekonfigurování a sestavit všechny používané programy tak, aby začínaly na správné počáteční adrese. Jak správně přeložit program pro danou počáteční adresu je vysvětleno na příkladu v kapitole 9. Pokud není program správně přeložen, pak jeho spuštění selže nebo nebude správně pracovat.

6 Aplikační knihovna pro PC

V průběhu projektu byla k zavaděči vytvořena knihovna, která implementuje představený komunikační protokol a umožňuje tak snadnou komunikaci se zavaděčem. Knihovna je vytvořena pomocí objektového paradigmatu a jazyka C++ ve frameworku Qt verze 5.5.1, aby se zajistila snadná udržovatelnost a přenositelnost mezi různými operačními systémy. V následujících podkapitolách jsou popsány implementační detaily této knihovny.

6.1 Návrh knihovny

Diagram 6.1 zobrazuje hierarchii tříd v knihovně. Třídy v první úrovni (BootloaderComm, BootloaderFinder, CommInterface a CommSerialInfo) jsou přímo dostupné přes rozhraní knihovny aplikacím, které tuto knihovnu využívají. Třídy CommSerial a CommSocket jsou pak schované uvnitř knihovny a realizují komunikaci prostřednictvím odpovídajících rozhraní. Příloha B detailněji zachycuje vztahy mezi jednotlivými třídami a jejich atributy pomocí diagramu tříd.



6.1 Diagram dědičnosti tříd softwarové knihovny

Jak diagram 6.1 naznačuje, všechny hlavní třídy v knihovně jsou odvozeny od třídy QObject. Tato třída je základní třídou frameworku Qt, která zpřístupňuje odvozeným třídám funkcionalitu signálů a slotů. Tento koncept je v aplikační knihovně rozsáhle využit, protože se jedná o efektivní implementaci reakcí na události. Signál je definován jako deklarace metody s formálními parametry, která postrádá definici těla a slot je definován jako metoda třídy typu QObject s odpovídajícími formálními parametry. Tyto dva elementy lze propojit voláním statické metody *connect()* třídy QObject tak, že po aktivaci signálu pomocí klíčového slova *emit*, následovaného jménem signálu a odpovídajícími skutečnými parametry, se zavolají všechny dříve zaregistrované metody s dosazenými skutečnými parametry. Tento princip je velmi užitečný v případě, kdy se předávají data z nižších aplikačních vrstev do vyšších a kdy není žádoucí vzájemná závislost mezi těmito vrstvami. Signály a sloty tak vlastně implementují jistou formu návrhového vzoru *Pozorovatel*. Příklad použití v knihovně je zobrazen úsekem kódu na obrázku 6.2.

```
class CommSocket : public CommInterface
{
public:
    …
    bool CommSocket::connect()
    {
        m_socket = new QTcpSocket();
        /* Propojení signálu a slotu */
        QObject::connect( m_socket, SIGNAL(connected()),
                          this, SLOT(tcp_connected()));
        …
        m_socket->connectToHost( dst_addr, REMOTE_PORT );
        …
    }

private slots:
    …
    void tcp_connected()
    {
        if(m_state != state )
        {
            m_state = state;
            /* Aktivace signálu */
            emit connectionStateChanged( state );
        }
    (
};
```

6.2 Ukázka použití signálů a slotů ve třídě CommSocket

Třída CommSocket může díky signálům a slotům například jednoduše reagovat na události vzniklé v TCP/IP zásobníku operačního systému. V metodě *connect()* třídy CommSocket je při zahájení připojování k zavaděči vhodně zaregistrována funkce *tcp_connected()* pro odchycení události navázání spojení, která vzniká ve třídě QTcpSocket. Až dojde k ustanovení TCP spojení mezi zavaděčem a knihovnou, bude vyvolán signál *connected()* třídy QTcpSocket, který způsobí invokaci zaregistrované metody *tcp_connected()*. Pokud se stav změnil, tak tato třída nastaví nový vnitřní stav spojení do proměnné *m_state* a upozorní vyšší vrstvy o změně stavu stejným způsobem – aktivuje signál *connectionStateChanged()* s parametrem odpovídajícím novému stavu spojení. Díky asynchronní povaze signálů může aplikace/knihovna využívat stávající kontext provádění i k jiným činnostem a překrýt tak blokující operace.

Použití signálů a slotů má však i své nevýhody. Především jejich použití vede na složitější kód, jelikož slot může být vyvolán kdykoliv během zpracování hlavní smyčky programu reprezentovaného metodou *exec()* třídy QCoreApplication, respektive QEventLoop. To implikuje ukládání jedné či více stavových proměnných, na základě kterých bude rozhodováno o dalším toku programu při jejich invokaci. Ladění takového programu či knihovny je velmi obtížné, protože s přibývajícím počtem stavových proměnných narůstá počet kombinací všech stavů a vzniká stavová exploze. Původní myšlenka využití pouze signálů a slotů pro řízení komunikace a přenosu dat mezi zavaděčem a knihovnou se z tohoto důvodu nakonec neosvědčila a jako mnohem jednodušší se ukázalo použití

sekvenčního kódu s blokujícími operacemi. Uplatnění signálů a slotů se tak přesunulo spíše do rozhraní knihovny, kde může být snadno použito pro interakci s grafickým uživatelským rozhraním, viz kapitola 8.

6.1.1 Třída BootloaderComm

BootloaderComm je základní třídou knihovny, která abstrahuje funkce a atributy skutečného zavaděče pomocí svých metod.

Před prvním použitím třídy je nutné nastavit požadované komunikační rozhraní zasláním zprávy *setInterface()*. Jejím parametrem je buď textový řetězec popisující rozhraní (např. komunikační port COM12, /dev/ttyACM či IPv4 adresa 192.168.137.5), nebo popisovač sériového rozhraní třídy CommSerialInfo. Na základě této identifikace se pomocí statické tovární metody *makeCommInterface()* vytvoří objekt třídy CommInterface, který realizuje komunikaci buď prostřednictvím sériového, nebo Ethernetového rozhraní. Nastavením komunikačního objektu se však ještě nevytvoří komunikační kanál. Ten je nutné otevřít zasláním zprávy *connect()* instanci třídy BootloaderComm. Toto volání je delegováno nastavenému komunikačnímu objektu, který jej realizuje – u sériového rozhraní dojde k výměně synchronizačních zpráv mezi zavaděčem a knihovnou, u ethernetového rozhraní se vytvoří nové TCP spojení. Následně je již možné komunikační objekt využívat pro přenos zpráv. Odpojení rozhraní po dokončení činnosti se provede pomocí metody *disconnect()*.

Třída BootloaderComm dále zajišťuje zpracování datové části paketů komunikačního protokolu a uchovává hodnoty jednotlivých atributů. Pro každý atribut je vytvořeno několik typů metod:

- *get* – veřejná metoda, která vrací lokálně uloženou hodnotu atributu.
- *set* – veřejná metoda, která provede nastavení atributu v zavaděči na požadovanou hodnotu. Tato metoda je vytvořena pouze pro zapisovatelné atributy.
- *read* – chráněná metoda, která provede vyčtení hodnoty atributu ze zavaděče a uložení do lokální kopie

Metody *read* a *set* dále volají vnitřní funkce *readAttribute()*, respektive *writeAttribute()* pro čtení a zápis hodnoty atributu. Tyto metody dále využívají společnou privátní metodu *execCommand()*, která slouží k zaslání jednoho paketu skrze zvolené komunikační rozhraní směrem

6.3 Graf volání metody readAttribute

k zavaděči. Obrázek 6.3 ilustruje graf volání funkce *readAttribute()*. Tato funkce připraví datovou část paketu s příkazem GET_ATTR (viz příloha C.2) a parametrem odpovídajícím číslu požadovaného atributu. Takto připravená zpráva je předána metodě *execCommand()*, která ji přepošle objektu zajišťujícímu komunikaci skrze zvolené komunikační rozhraní předáním zprávy *writeMessage()*. Odpověď je následně získána pomocí metody *readResponse()* a správnost provedení je určena hodnotou návratového kódu obou těchto metod. Efektivně se tak využívá princip dědičnosti – třída BootloaderComm nepotřebuje znát detaily odeslání zprávy, jako je obalení zprávy správnými hlavičkami a patičkami níže postavených protokolů, výpočet CRC apod. To je zajištěno v podřazených třídách.

Metodu *execCommand()* využívají také další veřejné metody realizující jednotlivé funkce zavaděče:

- *uploadFirmware()* – metoda pro nahrání souboru s firmware do flash paměti MCU. Parametrem je cesta k nahrávanému souboru.
- *downloadFirmware()* – metoda pro stažení aktuálního firmware z MCU. Parametrem je cesta k ukládanému souboru.
- *runApplication()* – metoda pro nastavení příznaku pro spuštění programu po odpojení komunikačního rozhraní.
- *saveConfiguration()* – metoda pro zaslání příkazu pro uložení stávající konfigurace zavaděče do flash paměti MCU.

Metoda *uploadFirmware()* podporuje pouze přenos binárních souborů, jejichž obsah je přímo nakopírován do flash paměti MCU. Pokud je parametrem specifikován například soubor s příponou .hex nebo .s19, které obvykle generují integrovaná vývojová prostředí pro vestavené aplikace, pak se tento soubor sice do flash paměti MCU uloží, ale jeho spuštění selže. Proto by aplikace používající knihovnu měly takový vstupní soubor nejprve převést do binárního formátu např. pomocí utility *objcopy* (je součástí sady nástrojů GNU tools for ARM®), nebo generovat binární soubor přímo ve vývojovém prostředí, pokud je to možné. Druhý způsob je podrobněji popsán v kapitole 9.

6.1.2 Třída BootloaderFinder

Tato třída byla implementována za účelem snadnější detekce dostupných zavaděčů. K tomu účelu spravuje následující 3 seznamy:

1) Seznam IP adres nalezených zavaděčů na lokálním segmentu sítě Ethernet
2) Seznam zavaděčů připojených skrze sériová rozhraní USB či RS-232.
3) Seznam připojených ladících rozhraní OSBDM

První seznam je aktualizován na základě odpovědí obdržených od síťových zavaděčů. Třída implementuje UDP klienta detekčního protokolu popsaného v kapitole 5.5.2. Po zahájení vyhledávání

je každou 1 sekundu odeslán detekční paket. Na základě odpovědí je upravován seznam detekovaných zavaděčů a notifikace o změně seznamu je provedena prostřednictvím signálu *ethernetDeviceListChanged()*.

Druhý seznam obsahuje zavaděče detekované pomocí sériových rozhraní. Během aktivního vyhledávání je seznam aktualizován jednou za sekundu a informace o změnách seznamu jsou předávány signálem *serialDeviceListChanged()*. Seznam může být aktualizován také jednorázově zasláním zprávy *updateSerialDeviceList()*. Tento seznam je tvořen objekty třídy CommSerialInfo, která slouží k popisu jednotlivých COM portů počítače. Na základě známé kombinace identifikátorů VID:PID u zavaděče (USB) a u sériového rozhraní RS-232 zprostředkovaného pomocí OSBDM jsou vybrány pouze ty COM porty, u kterých lze očekávat možnost komunikace se zavaděčem. V případě OSBDM sériového kanálu však nelze bez předchozího navázání komunikace zjistit, zda na straně MCU aktuálně běží zavaděč či nikoliv. Z principu funkce USB rozhraní je také velmi obtížné zjištění příslušnosti připojeného OSBDM rozhraní k samotné platformě Minerva, resp. k dalším USB komponentám (VNC2, MK60, USB hub) nacházejícím se na téže fyzické desce tištěných spojů platformy Minerva. Z těchto důvodů může seznam sériových rozhraní obsahovat i ty COM porty, pomocí kterých není možné dočasně či trvale navázat komunikaci se zavaděčem a pokusy o připojení k zavaděči pomocí takových portů skončí neúspěšně.

Třetí seznam obsahuje detekovaná ladící rozhraní OSBDM, která jsou vyhledána pomocí externí knihovny OSBDM-JM60. Výsledkem vyhledávání je pouze počet nalezených ladících rozhraní. Navázání komunikace s daným rozhraním se děje na základě indexu v rozsahu od 0 do počtu nalezených rozhraní a odehrává se během požadavku na reset mikrokontroléru MK60 z aplikační knihovny. Stejně jako v případě sériového kanálu OSBDM se i zde objevuje problém vytvoření asociace mezi detekovanými ladícími rozhraními a připojenými platformami Minerva. Proto je nutné při každém uživatelském resetu zadat index rozhraní OSBDM, které má být využito.

6.2 Rychlost přenosu souboru

Rychlost programování je jednou z hlavních nevýhod při použití rozhraní OSBDM v kombinaci s vývojovým prostředím KDS a jednou z příčin vzniku nového zavaděče. Proto následuje srovnání rychlosti přenosu testovacího firmware o velikosti 64 KB pomocí jednotlivých rozhraní zavaděče. Rychlost přenosu je získána následujícím vztahem.

$$v = \frac{velikost\ souboru}{celková\ doba\ přenosu} \tag{6.1}$$

Jedná se tedy pouze o orientační rychlost přenosu dat, která zahrnuje také režii komunikačního protokolu. Soubor se přenáší po částech o velikosti 2 KB, přičemž na každou část přenášenou do zavaděče činí režie protokolu 7 bajtů a režie odpovědi je 8 bajtů. Součet režií je tedy 15 bajtů, což

	Rozhraní			
	Ethernet	USB	RS-232	SDHC
Nahrávání	23,91 KB/s	21,66 KB/s	7,07 KB/s	37,56 KB/s
Stahování	426,37 KB/s	118,46 KB/s	11,08 KB/s	-

6-1 Rychlost nahrávání a stahování uživatelského firmware

odpovídá zhruba 1% z počtu přenesených dat v jednom paketu. Režie je stejná také pro opačný směr přenosu dat.

Větší problém představuje zpoždění na straně zavaděče (zpracování dat trvá nenulovou dobu), kvůli kterému klesá rychlost přenosu velmi razantně. To lze pozorovat hlavně při nahrávání souboru do MCU – zavaděč nemůže při programování flash paměti MCU provádět žádnou jinou činnost, takže rychlost přenosu souboru je zároveň omezena rychlostí zápisu do flash paměti.

Tabulka 6-1 zobrazuje naměřené rychlosti přenosu souboru oběma směry. Hodnoty jsou průměrem z 5 měření. Je možné pozorovat minimální rozdíl v rychlosti nahrávání souboru při použití rozhraní USB a Ethernet – jejich rychlost mnohonásobně přesahuje rychlost zpracování přijatých dat v zavaděči. Naopak rychlost přenosu pozorovaná u rozhraní RS-232 ve směru od zavaděče odpovídá nastavení 115200 baudů, 8 datových bitů, 1 stop bit, bez paritních bitů.

Zajímavým zjištěním je přenosová rychlost dosažená při nahrávání souboru přes rozhraní SDHC, která je téměř dvojnásobná oproti rozhraní Ethernet a USB. Způsob použití tohoto rozhraní je však značně odlišný od ostatních, protože aplikace řídící přenos souboru je přímo součástí zavaděče. Data souboru jsou tedy z SD karty přenášena pomocí DMA kanálu do vnitřní RAM paměti MCU, odkud jsou následně zapisována do flash paměti. Pro měření byla použita SD karta PQI s kapacitou 128 MB nespadající do žádné z SD rychlostních tříd a rychlost SDHC rozhraní byla nastavena na 8 MHz.

7 Konzolová aplikace

Pro snadnou práci se zavaděčem byla vytvořena jednoduchá konzolová aplikace BootConfig na bázi aplikační knihovny popsané v předchozí kapitole. Aplikace zpřístupňuje všechny funkce dostupné v aplikační knihovně a automatizuje proces přístupu k zavaděči z PC. Mezi klíčové vlastnosti této aplikace patří:

- Jednoduchá implementace
- Snadná rozšiřitelnost
- Snadná přenositelnost mezi systémy Windows a Linux
- Provádění sekvence transakcí bez uživatelského zásahu
- Podpora zdrojových souborů s firmware ve formátu .bin.
- Automatická detekce zavaděčů na různých rozhraních

Aplikace je implementována pomocí jazyka C++ a knihovny Qt verze 5.5.1. Následující podkapitola ve stručnosti popisuje návrh aplikace. Příklad použití konzolové aplikace pro nahrání uživatelského programu se nachází v kapitole 9.

7.1 Struktura aplikace

Návrh konzolové aplikace je poměrně přímočarý, protože veškerý kód běží synchronně pouze v jednom vlákně. Využívá se tak blokujících operací v aplikační knihovně, které jsou volány z konzolové aplikace podle zadaných argumentů příkazové řádky. Schéma hlavní funkce konzolové aplikace je znázorněno úsekem kódu na obrázku 7.1.

```
int main(int argc, char *argv[])
{
    ...
    loadLoggingRules();         // Načtení pravidel pro výpisy
    parserCommandLine();        // Zpracování argumentů
    resetMcu();                 // Resetování MCU (je-li požadováno)
    detectBootloaders();        // Automatická detekce zavaděčů
    if( m_options.cmds.count() > 0 )
    {
        connectToBootloader();          // Připojení k zavaděči
        processCommands();              // Zpracování příkazů
        disconnectFromBootloader();     // Odpojení od zavaděče
    }
    return 0;
}
```

7.1 Hlavní funkce konzolové aplikace

První krok je načtení pravidel pro výpis zpráv do terminálu. Pravidla jsou uložena v souboru *qtlogging.ini*, který je součástí souboru se zdroji (tzv. resource soubor). Soubor s pravidly slouží k odfiltrování ladících zpráv a umožňuje selektivní kontrolu nad jednotlivými kategoriemi výpisů z aplikační knihovny. Obsah souboru vypadá následovně:

```
[Rules]
*.debug=false                // Zakáže ladící výpisy všech kategorií
bl.finder.device.info=false // Zakáže výpis informace o nalezeném zavaděči
```

7.1 Příklad obsahu souboru qtlogging.ini

Druhým krokem je zpracování argumentů příkazového řádku a odpovídající naplnění struktury *m_options*, jež je dále používána pro řízení toku programu. Nejdůležitější položkou struktury je seznam *cmds*, který obsahuje pole příkazů, které se provedou po připojení k zavaděči. Příkazy se vykonávají v pořadí, v jakém jsou specifikovány pomocí odpovídajících argumentů na příkazovém řádku. Popis jednotlivých argumentů je součástí nápovědy programu.

Třetím krokem je reset cílového MCU pomocí rozhraní OSBDM. Není-li definováno jinak, provede se vždy reset do zavaděče, protože je očekávána následná komunikace se zavaděčem. Pokud je k PC připojeno více kusů platformy Minerva pomocí rozhraní USB, pak je u volby *reset* potřeba správně zvolit použité OSBDM rozhraní, aby proběhl reset vybraného MCU.

Ve čtvrtém kroku může probíhat automatická detekce zavaděčů, pokud byla požadována. Nastavení se provádí pomocí volby *interface*. Výchozí doba vyhledávání je 10 sekund a lze ji zkrátit nebo prodloužit pomocí volby *autodetect-timeout*. Tabulka 7-1 zachycuje možnosti nastavení komunikačního rozhraní. Při použití automatické detekce se na základě omezujících podmínek

--interface=…	**Popis**
Název sériového rozhraní (např. COM9)	Použije přímo zvolené rozhraní, detekce se neprovádí.
IPv4 adresa	
autodetect	Detekce na všech rozhraních
autodetect-ethernet	Detekce pouze na rozhraní Ethernet (LAN)
autodetect-serial	Detekce na rozhraních USB a RS232
autodetect-usb	Detekce pouze na rozhraní USB
autodetect-rs232	Detekce pouze na rozhraní RS-232 (OSBDM)

7-1 Tabulka parametru volby *interface*

použije první nalezený zavaděč v pořadí. Automatická detekce tedy funguje spolehlivě pouze v případě, kdy je k PC připojen pouze jediný zavaděč. Jinak je lepší pomocí volby *interface* explicitně specifikovat požadované rozhraní.

Pokud byl zadán nějaký příkaz pro zavaděč, pak proběhne také nastavení komunikačního rozhraní a vytvoření komunikačního kanálu mezi aplikací a zavaděčem. Dále jsou provedeny všechny požadované akce, jimž odpovídají jednotlivá volání aplikační knihovny. Poslední krokem je ukončení spojení se zavaděčem.

8 Grafická aplikace

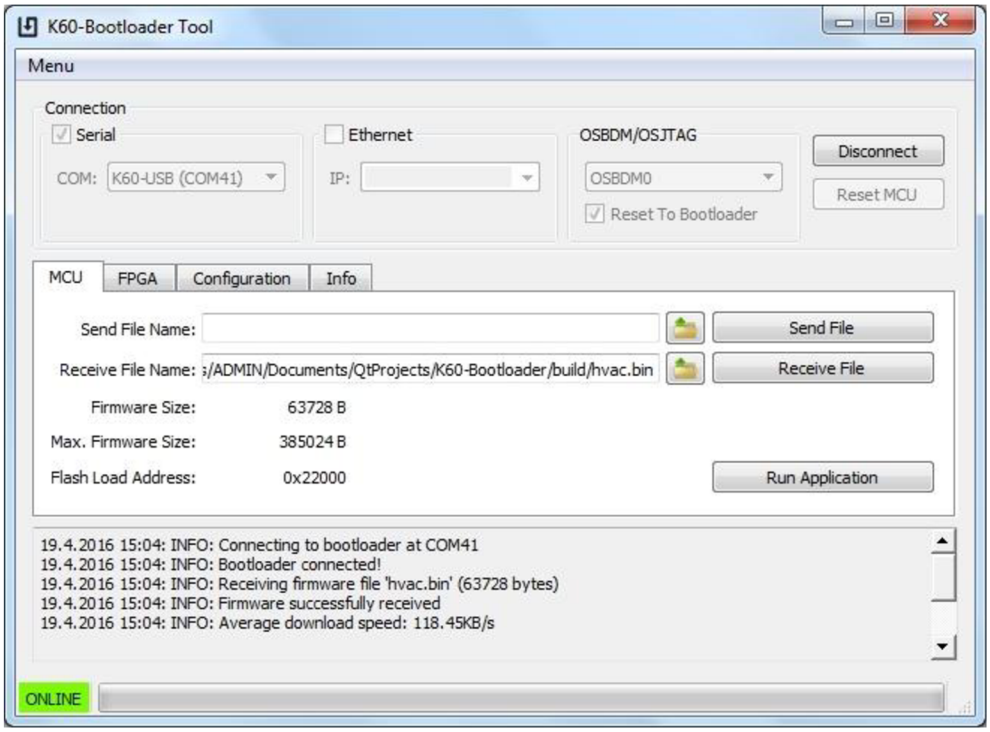
Pro ještě snadnější práci se zavaděčem byla dále vytvořena grafická aplikace s názvem K60-Bootloader Tool taktéž využívající aplikační knihovnu. Uživatel si tak může vybrat, jestli mu více vyhovuje grafické nebo konzolové prostředí pro práci se zavaděčem. Grafická aplikace vyniká především intuitivním ovládáním a snadností použití.

8.1 Hlavní okno

Hlavní okno aplikace je znázorněno na obrázku 8.1 a je rozděleno do 3 částí.

V horní části se nachází parametry připojení k zavaděči a jejich ovládací prvky. Je možné vybrat buď sériové rozhraní, nebo Ethernetové rozhraní. V rámci sériového rozhraní je nabídnut výběr mezi sériovými porty asociovanými se zavaděčem. U Ethernetového rozhraní je pak možné zadat IP adresu, na které se nachází zavaděč, nebo ji lze vybrat ze seznamu adres zavaděčů detekovaných na lokální síti. Připojení k zavaděči se provede pomocí tlačítka *„Connect“*, které se po úspěšném připojení změní na tlačítko *„Disconnect“* sloužící k odpojení od zavaděče.

V odpojeném stavu je možné provést ruční reset MCU pomocí tlačítka *„Reset MCU“*. V tom případě se provede reset pomocí vybraného OSBDM rozhraní a podle zatržení volby *„Reset To*



8.1 Úvodní obrazovka aplikace K60-Bootloader Tool

Bootloader" se do MCU zapíše patřičný příznak pro spuštění zavaděče či nikoliv.

Ve střední části se nachází několik záložek s oddělenými funkčnostmi. Záložka MCU slouží k nahrávání a stahování firmware z MCU. Záložka FPGA poskytuje uživatelské rozhraní k programování konfigurační flash paměti FPGA na platformě Minerva pomocí frameworku navrženého Petrem Buchtou [13]. Záložka Configuration slouží k nastavení a uložení parametrů zavaděče. Na záložce Info se nacházejí dodatečné informace o zavaděči jako je číslo verze a podporovaná rozhraní.

Ve spodní části se nachází informační a stavová obrazovka, ve které se zobrazují hlášení a chyby vzniklé za běhu programu. Výpisy se nastavují stejně jako v případě konzolové aplikace pomocí konfiguračního souboru *qtlogging.ini*. Ve stavovém řádku aplikace se zobrazuje stav připojení a průběh stávající operace.

8.2 Struktura aplikace

Z implementačního hlediska se jedná o objektovou aplikaci vytvořenou v jazyce C++ s použitím frameworku Qt 5.5.1. Grafické prostředí bylo vytvořeno v nástroji QtCreator 3.5.1 a je částečně variabilní, co se týká rozměrů okna.

Jelikož se jedná o grafickou aplikaci, tak je běh programu rozdělen do dvou vláken – hlavní vlákno se stará o vykreslování uživatelského prostředí a vedlejší vlákno provádí jak blokující operace třídy BootloaderComm, tak vyhledávání zavaděčů pomocí instance třídy BootloaderFinder. Díky tomu nedochází k blokování uživatelského prostředí a ztrátě uživatelského komfortu.

Pro komunikaci mezi vlákny je použito mechanizmu signálů a slotů. Ty fungují i ve více-vláknovém prostředí díky mechanizmu zvanému QueuedConnection. Jedná se o typ spojení mezi dvěma objekty, při kterém jsou data předávána skrze zabezpečenou frontu událostí. Tyto dva objekty se poté mohou nacházet v různých vláknech a při tom bude zabezpečeno, že k zaslání signálu dojde ve vlákně odesílatele a vykonání odpovídajícího slotu s předanými parametry se provede ve vlákně příjemce. Důležitou podmínkou funkčnosti je vlastní cyklus pro obsluhu událostí reprezentovaný objektem třídy QEventLoop nebo QCoreApplication v každém vlákně implementujícím objekty se sloty. V případě vedlejšího vlákna je již požadovaný cyklus součástí třídy QThread, který zapouzdřuje vedlejší vlákno.

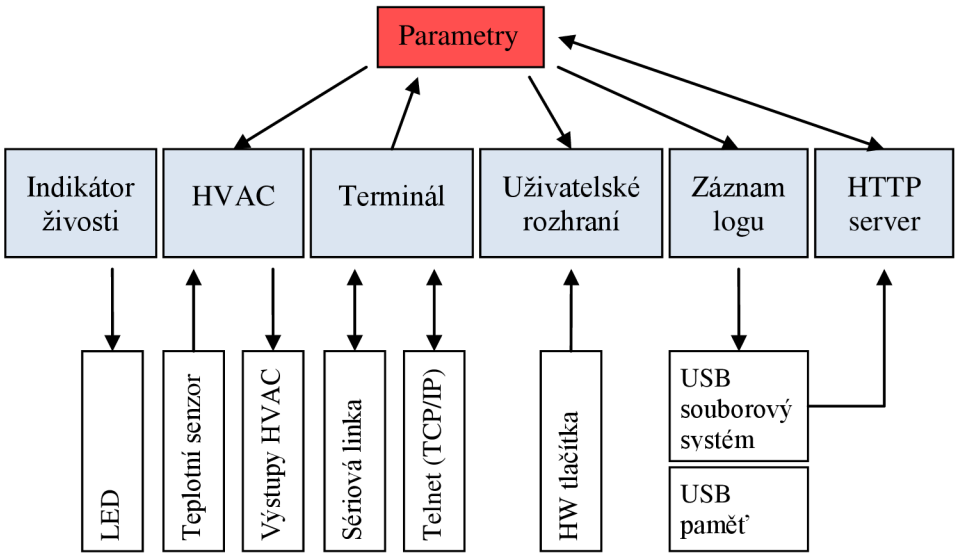
9 Demonstrační aplikace

Pro demonstraci správné funkčnosti zavaděče a ukázku jeho použití byla vybrána existující aplikace WEB HVAC, připravená pro platformu Minerva od společnosti Freescale. V této kapitole se nachází stručný popis této aplikace, postup při překladu aplikace pro použití se zavaděčem a ukázka možných způsobů nahrání aplikace do MCU.

9.1 Popis aplikace

Hlavní funkcí demonstrační aplikace WEB HVAC je řízení systému vytápění, odvětrávání a klimatizace v bytových jednotkách. Na základě informací z teplotního senzoru aplikace ovládá tři dvoustavové výstupy, jimiž se spíná topení, ventilátor a klimatizační jednotka. Sledování stavu a nastavení parametrů je možné pomocí terminálu přístupného přes sériové rozhraní či telnet. Součástí aplikace je také ukládání teplotního logu na externí USB paměťové médium [14].

Hlavním cílem aplikace je ukázka použití operačního systému reálného času MQX od společnosti Freescale. Jednotlivé úlohy systému jsou rozděleny do 6 hlavních vláken, které běží nezávisle na sobě a komunikují spolu pomocí synchronizačních prostředků – semaforů a schránek pro předávání zpráv. Tyto úlohy jsou modře znázorněny na obrázku 9.1. Dalším významným cílem je ukázka použití periferií USB a Ethernet v kombinaci se systémem MQX. Pro každou z těchto periferií je vytvořeno vlákno, které implementuje příslušný protokolový zásobník. V případě periferie USB se jedná o *Freescale USB Stack*, který je také použit v zavaděči. U rozhraní Ethernet je využit TCP/IP protokolový zásobník *RTCS* taktéž od firmy Freescale.



9.1 Schéma úloh demonstrační aplikace a souvisejících činností. Převzato a upraveno z [14].

9.2 Příprava aplikací

9.2.1 Příprava zavaděče

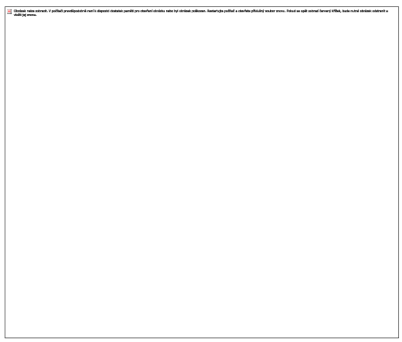
Prvním krokem je zavedení zavaděče do mikrokontroléru. S ohledem na zvažované rozhraní je vhodné zvolit správnou překladovou konfiguraci zavaděče. Pro demonstraci funkčnosti je ukázáno nahrávání demonstrační aplikace pomocí rozhraní USB, Ethernet a SDHC, takže byla vybrána překladová konfigurace 6 podle tabulky 5-1. Tato konfigurace určuje počáteční adresu demonstračního programu v paměti mikrokontroléru na adrese 0x15000, což bude potřeba při překladu aplikace zajistit.

9.2.2 Vytvoření projektu demonstrační aplikace

Demonstrační aplikace je součástí balíku aplikací a ovladačů dodaných k platformě Minerva firmou Freescale. Projekt této aplikace pro prostředí KDS je již vytvořen, stačí jej tedy v rámci uživatelského pracovního prostoru propojit se souvisejícími projekty pomocí funkce File → Import. Těmi jsou:

1) PSP – Port operačního systému MQX na mikrokontrolér MK60
2) BSP – Projekt obsahující ovladače k periferiím MCU využitým na platformě Minerva
3) MFS – Souborový systém FAT
4) RTCS – Protokolový zásobník TCP/IP
5) Shell – Projekt obsahující implementaci generického terminálu pro MCU
6) USB – Protokolový zásobník USB s podporou funkce USB hosta

Stav pracovního prostoru (workspace) po importování všech projektů je znázorněn na obrázku 9.2:



9.2 Importování závislostí projektu WEB HVAC v prostředí KDS

9.2.3 Překlad demonstrační aplikace

Při překladu demonstrační aplikace nebo jiné uživatelské aplikace, která bude nahrávána zavaděčem je potřeba zjistit její sestavení od správné počáteční adresy. Detailní postup překladu v prostředí KDS je popsán v souboru *src_demo/readme.txt* na přiloženém na CD. Obecně je potřeba definovat vhodný

sestavovací skript tak, aby byl uživatelský program sestaven s ohledem na omezení kladená zavaděčem -v tomto konkrétním případě je počáteční adresa 0x15000 a maximální velikost programu 428 KB.

Příklad potřebných úprav sestavovacího skriptu se nachází na obrázku 9.3. Soubor *intflash.ld* je původní sestavovací skript, který nepočítá s využitím zavaděče. Soubor *intflash_bootloader.ld* je pak upravený soubor, ve kterém se změnily počáteční adresy sekcí *vectorrom*, *cfmprotrom* a *rom* tak, aby respektovaly požadavky zavaděče.

intflash.ld

```
11 MEMORY
12 {
13     vectorrom   (RX): ORIGIN = 0x00000000, LENGTH = 0x00000400
14     cfmprotrom  (R): ORIGIN = 0x00000400, LENGTH = 0x00000020
15     rom         (RX): ORIGIN = 0x00000420, LENGTH = 0x0007FBE0  /* Code + C
16     ram         (RW): ORIGIN = 0x1FFF0000, LENGTH = 0x00020000  /* SRAM - R
17
18     /* kernel space starts after RAM variables (Location of MQX Kernel data
19     end_of_kd   (RW): ORIGIN = 0x2000FFF0, LENGTH = 0x00000000
20
21     /* Boot stack reused by MQX Kernel data */
22     bstack      (RW): ORIGIN = 0x2000FA00, LENGTH = 0x00000200  /* Boot sta
23     end_bstack  (RW): ORIGIN = 0x2000FC00, LENGTH = 0x00000000  /* Boot sta
24 }
25
```

intflash_bootloader.ld

```
11 MEMORY
12 {
13     vectorrom   (RX): ORIGIN = 0x00015000, LENGTH = 0x00000400
14     cfmprotrom  (R): ORIGIN = 0x00015400, LENGTH = 0x00000020
15     rom         (RX): ORIGIN = 0x00015420, LENGTH = 0x0006ABE0  /* Code + C
16     ram         (RW): ORIGIN = 0x1FFF0000, LENGTH = 0x00020000  /* SRAM - R
17
18     /* kernel space starts after RAM variables (Location of MQX Kernel data
19     end_of_kd   (RW): ORIGIN = 0x2000FFF0, LENGTH = 0x00000000
20
21     /* Boot stack reused by MQX Kernel data */
22     bstack      (RW): ORIGIN = 0x2000FA00, LENGTH = 0x00000200  /* Boot sta
23     end_bstack  (RW): ORIGIN = 0x2000FC00, LENGTH = 0x00000000  /* Boot sta
24 }
25
```

9.3 Úprava počáteční adresy programu v sestavovacím skriptu

Při překladu je potřeba věnovat zvláštní pozornost paměťové sekci *cfmprotrom*. Ta se standardně mapuje do konfigurační oblasti flash paměti na MK60 a obsahuje informace týkající se zabezpečení jednotlivých bloků flash paměti. Relokací na jinou adresu přestává tato sekce plnit svůj účel. Z toho plyne omezení na uživatelské programy, které nemohou používat zabezpečení flash paměti bez přídavné podpory v zavaděči. Zde se tedy naskýtá prostor pro další možné vylepšení zavaděče, který by mohl tuto sekci uživatelského programu překopírovat do konfiguračních registrů flash paměti a tím provést samotné zabezpečení ještě před spuštěním uživatelského programu.

```
Memory Configuration

Name             Origin             Length             Attributes
vectorrom        0x00015000         0x00000400         xr
cfmprotrom       0x00015400         0x00000020         r
rom              0x00015420         0x0007fbe0         xr
ram              0x1fff0000         0x00020000         rw
end_of_kd        0x2000fff0         0x00000000         rw
bstack           0x2000fa00         0x00000200         rw
end_bstack       0x2000fc00         0x00000000         rw
*default*        0x00000000         0xffffffff
```

9.4 Mapování paměťových sekcí na fyzické adresy MK60. Počáteční adresa programu je 0x15000

Správné sestavení přeloženého programu je vhodné zkontrolovat pomocí mapovacího souboru, který je možné nechat vygenerovat spolu s překladem. Mapování jednotlivých paměťových sekcí na fyzické adresy by mělo odpovídat obrázku 9.4.

9.3 Spuštění aplikace

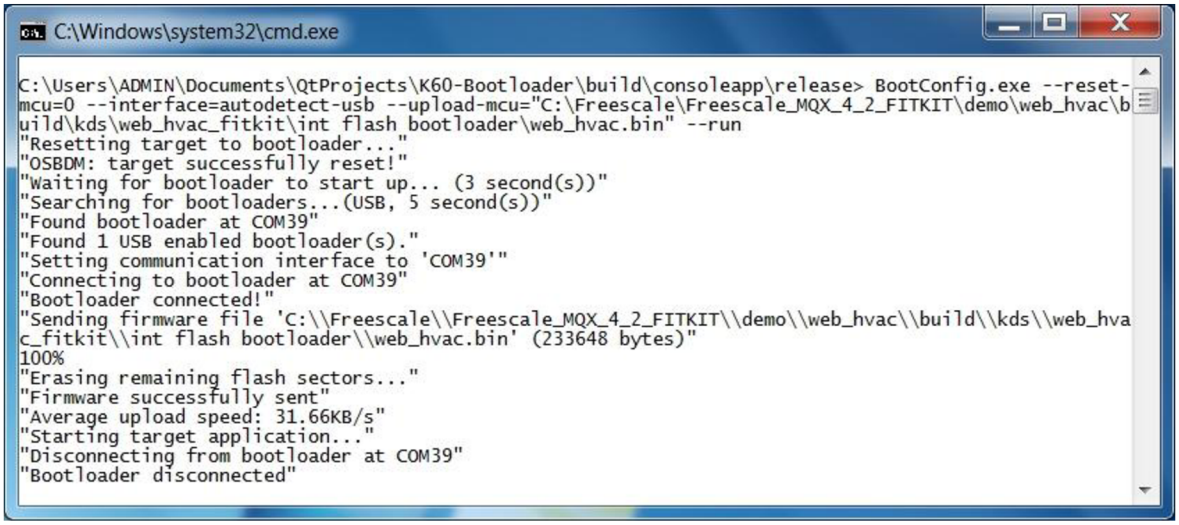
9.3.1 Spuštění z konzolové aplikace

Nahrání uživatelského programu do MK60 připojeného pomocí USB rozhraní se s využitím konzolové aplikace nejjednodušeji provede spuštěním s následujícími parametry:

BootConfig.exe --reset-mcu=0 --interface=autodetect-usb [--autodetect-timeout=...]
--upload-mcu=<cesta k souboru web_hvac.bin> --run

Parametr *reset-mcu* zajistí reset mikrokontroléru do zavaděče. Hodnota předávaná s tímto parametrem udává číslo rozhraní OSBDM v systému. Pokud je takové rozhraní pouze jediné, pak je vždy specifikováno hodnotou 0. V opačném případě je pomocí této hodnoty určeno konkrétní rozhraní OSBDM pro provedení resetu.

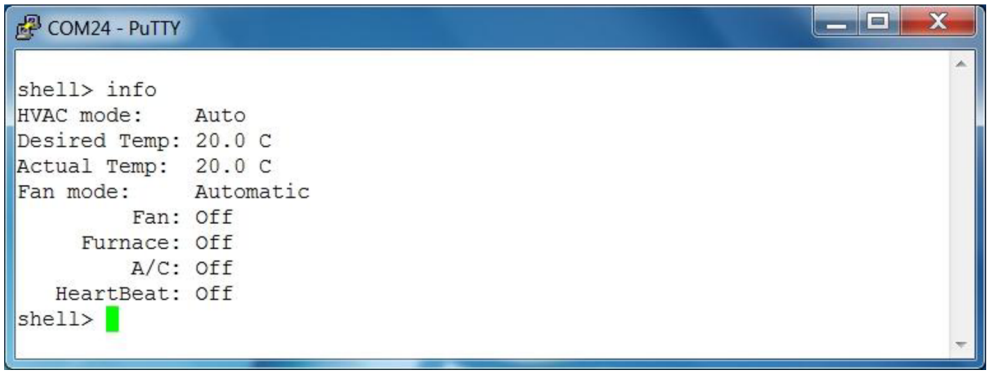
Druhý parametr určuje použité rozhraní. Jako ukázkové rozhraní bylo zvoleno rozhraní USB, avšak v případě volby Ethernetového rozhraní může být vhodné zadat také parametr *autodetect-timeout* a tím nastavit dobu potřebnou k vyhledání zavaděče na lokální síti.



9.5 Snímek obrazovky zachycující průběh nahrávání uživatelské aplikace pomocí konzolové aplikace

Třetí parametr udává absolutní nebo relativní cestu k souboru web_hvac.bin. Poslední parametr říká, že po nahrání aplikace má dojít k jejímu spuštění. Výstup z konzolové aplikace po spuštění příkazu je zobrazen na obrázku 9.5.

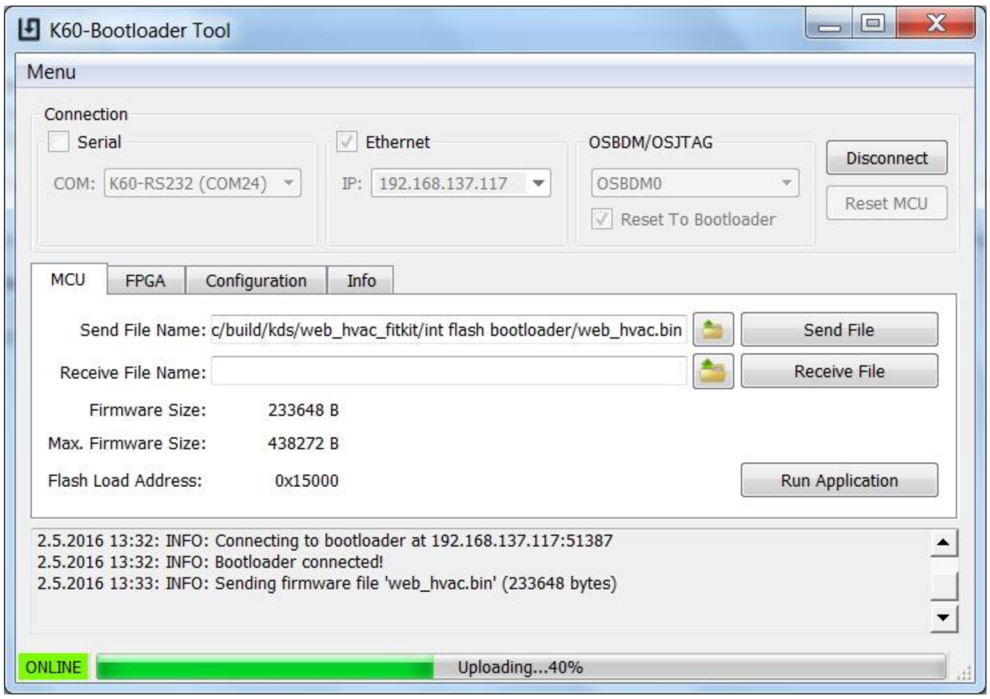
Po odpojení konzolové aplikace od zavaděče došlo k resetu MCU a spuštění aplikace WEB HVAC, k jejímuž terminálu bylo následně možné se připojit například pomocí aplikace Putty s využitím sériového kanálu rozhraní OSBDM. Výstup z terminálu je zobrazen na obrázku 9.6.



9.6 Připojení k terminálu aplikace WEB HVAC pomocí aplikace Putty

9.3.2 Spuštění z grafické aplikace

Nahrávání uživatelského programu je zachyceno na obrázku 9.7. V tomto případě bylo použito pro připojení k zavaděči rozhraní Ethernet, přičemž PC a platforma Minerva byly propojeny napřímo pomocí UTP kabelu. Zavaděč měl dále povoleno získání adresy z DHCP serveru, který byl spuštěn na lokálním PC.



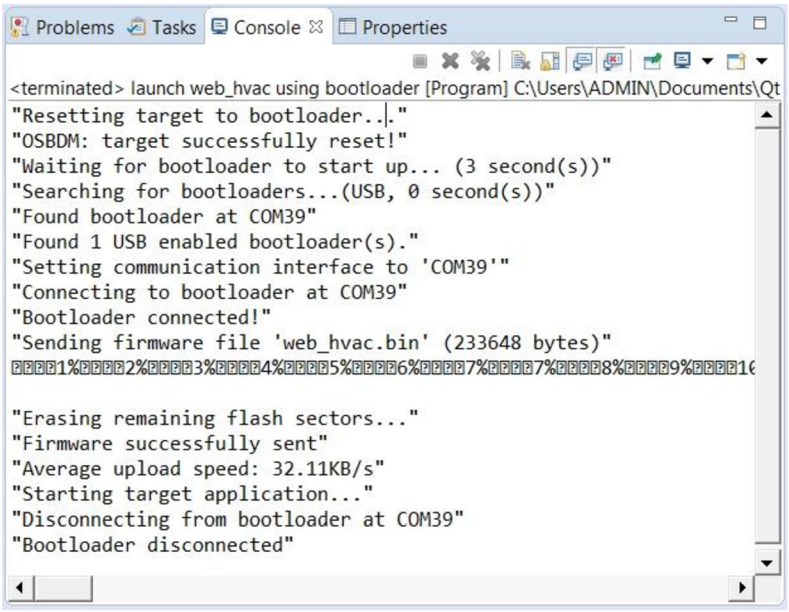
9.7 Snímek obrazovky zachycující průběh nahrávání uživatelské aplikace pomocí grafické aplikace

Největší rozdíl v práci mezi grafickou a konzolovou aplikaci je ten, že v případě grafické aplikace nejsou jednotlivé kroky nahrávání firmware příliš automatizovány a vázány na jiné kroky. Je tedy nutné nejprve vybrat správné rozhraní, dále vybrat nahrávaný soubor, kliknutím na tlačítko *„Send File“* daný soubor nahrát do MCU a následně spustit nahraný program pomocí tlačítka *„Run Application“*. Celý postup tedy zabere mnohem více kroků, než by tomu bylo u konzolové aplikace, ale uživatel má větší kontrolu nad tím, co se právě provádí a může jednotlivé kroky provádět odděleně.

9.3.3 Spuštění z prostředí KDS

Zajímavá možnost uplatnění zavaděče se skrývá v jeho propojení s vývojovým prostředím KDS. Díky tomu je možné pohodlně vyvíjet uživatelský program a zároveň rychle testovat jeho funkčnost přímo z vývojového prostředí. Řešení spočívá v nastavení cíle pro spuštění uživatelského programu pomocí externího nástroje – konzolové aplikace. Té jsou předány odpovídající parametry a zodpovědnost za nahrání a spuštění zkompilovaného programu. Výstup konzolové aplikace je zobrazen v interním terminálu KDS, jak je vidět na obrázku 9.8. Bohužel tento terminál nepodporuje všechny řídící znaky, takže zobrazení průběhu není stejné jako v případě systémového terminálového okna.

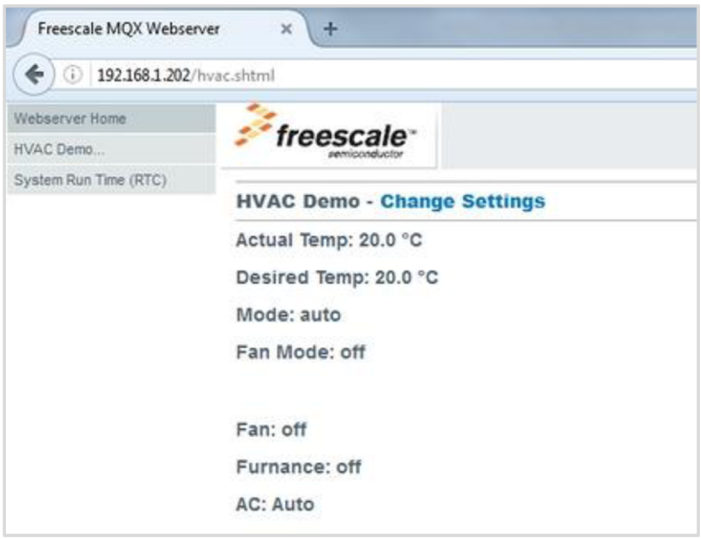
Zajímavé je srovnání rychlosti nahrávání programu WEB HVAC pomocí rozhraní OSBDM a zavaděče připojeného pomocí rozhraní USB-CDC. Zatímco v prvním případě nahrávání programu trvalo cca 50 sekund, tak pomocí zavaděče stejná akce trvala přibližně 10 sekund. Nutno podotknout, že v obou případech se jednalo o stejný optimalizovaný kód, který byl pouze v případě zavaděče relokován na adresu 0x15000. Druhý rozdíl je v tom, že prostředí KDS používá k programování soubor ve formátu ELF, zatímco zavaděč používá soubor .bin. I přesto tyto rozdíly zvolený příklad dokládá, že zavaděč se může v jistých situacích při práci s platformou Minerva jevit jako efektivnější ve srovnání s prostředím KDS a rozhraním OSBDM.



9.8 Průběh nahrávání z prostředí KDS

9.3.4 Spuštění z SD karty

Poslední možností jak nahrát uživatelský program do interní flash paměti MCU je pomocí SD karty. Binární soubor web_hvac.bin je přímo nakopírován do kořenového adresáře SD karty a přejmenován na MINERVA_FW.BIN. Takto pojmenovaný soubor je zavaděčem detekován a nahrán do uživatelské části flash paměti při jeho spuštění. Během nahrávání svítí dioda D9 s konstantní intenzitou, což signalizuje probíhající transakci. Po dokončení programování je možné nahranou aplikaci spustit stisknutím tlačítka SW6 a zkontrolovat její funkčnost například připojením se na webové rozhraní, které má v sobě implementováno – webová stránka je ve výchozím nastavení dostupná na IP adrese 192.168.1.202.



9.9 Webové rozhraní demonstrační aplikace WEB HVAC

V rámci práce byla také zvažována možnost speciálně pojmenovaného konfiguračního souboru v kořenovém adresáři SD karty, pomocí kterého by bylo možné specifikovat cestu k souboru s uživatelským programem. Díky tomu by mohlo být na SD kartě umístěných programů více a výběr mezi nimi by se prováděl pomocí konfiguračního souboru. Tato varianta byla sice implementována, ale po dohodě s vedoucím práce se nakonec ukázalo jako lepší řešení mít pouze jeden speciálně pojmenovaný soubor se samotným programem.

10 Závěr

Cílem této práce bylo vytvoření jednoduchého univerzálního zavaděče pro mikrokontrolér Kinetis K60, který by usnadnil nahrávání uživatelských aplikací do tohoto mikrokontroléru. Na začátku práce bylo analyzováno několik stávajících řešení, jejichž vlastnostmi je inspirován také nově navržený zavaděč. Široká škála podporovaných komunikačních rozhraní si vyžádala návrh nového komunikačního protokolu, který pokryl jak potřebu na konfiguraci zavaděče, tak i na přenos datových souborů. Navržený protokol dal následně kostru samotnému zavaděči, který se podařilo efektivně a relativně rychle vyvinout díky komponentám modulu Processor Expert z prostředí KDS.

V další části práce bylo potřeba vytvořit prostředky pro komunikaci se zavaděčem z PC. Jako vhodné řešení se ukázalo použití frameworku Qt, díky kterému je zajištěna snadná přenositelnost těchto prostředků mezi různými operačními systémy. V prvé řadě šlo o implementaci aplikační knihovny, která představuje rozhraní pro komunikaci se zavaděčem na straně PC. Nad touto knihovnou byly dále vystavěny dvě aplikace – BootConfig s konzolovým rozhraním a K60-Bootloader Tool s grafickým uživatelským rozhraním – které slouží k nahrávání uživatelských aplikací do MCU pomocí zavaděče.

V rámci práce se podařilo implementovat několik funkcí nad rámec zadání. Jedná se především o možnost zpětného stažení uživatelského programu z MCU do PC. Dále bylo provedeno propojení aplikační knihovny s knihovnou OSBDM-JM60, která aplikacím zpřístupnila funkci asynchronního resetu mikrokontroléru K60. Díky tomu je možné prakticky kdykoliv přerušit běh stávající aplikace a nahrát pomocí zavaděče aplikaci jinou. Významným rozšířením pak bylo zakomponování části diplomové práce Petra Buchty do zavaděče, což umožnilo provádět pomocí zavaděče také programování konfigurační flash paměti FPGA nacházející se na platformě Minerva.

Možností dalšího vývoje zavaděče se nabízí hned několik – do zavaděče by bylo možné integrovat rozhraní virtuálního JTAG adaptéru ať už formou samostatné služby běžící na rozhraní Ethernet nebo jako součást komunikačního protokolu. Ten by pak bylo možné využít k programování FPGA. Dále by bylo možné například rozšířit konfiguraci zavaděče o nastavení zabezpečení flash paměti, či vyhrazení části flash paměti pro účely nahraných aplikací apod. Další vylepšení by mohlo spočívat v možnosti konfigurování a spouštění zavaděče z uživatelské aplikace, což by umožnilo provádění automatických aktualizací této aplikace.

Nový zavaděč usnadňuje použití platformy Minerva především tím, že pro programování flash paměti mikrokontroléru zpřístupňuje různá rozhraní. Uživatel si tak může vybrat, které mu vyhovuje nejvíce a nemusí se spoléhat pouze na ladící rozhraní OSBDM. Navíc se s použitím zavaděče otevírají nové možnosti vývojové platformy Minerva – například možnost napojení na překladový systém nebo snadnější prezentace vlastních aplikací při výuce apod. Pevně věřím, že nový zavaděč pomůže tuto zajímavou platformu zpřístupnit co nejširšímu publiku, a to nejen z akademických řad.

11 Bibliografie

[1] FREESCALE SEMICONDUCTOR, INC. *K60 Sub-Family Reference Manual* [online]. 2011 [cit. 2015-12-28]. rev. 6. Dostupné z: http://cache.freescale.com/files/32bit/doc/ref_manual/K60P144M100SF2RM.pdf

[2] LOZANO, Alejandro a Alí PIñA. FREESCALE SEMICONDUCTORS, INC. *Ethernet Bootloader for MCU* [online]. 2011 [cit. 2015-12-28]. Dostupné z: http://cache.freescale.com/files/microcontrollers/doc/app_note/AN4367.pdf

[3] FREESCALE SEMICONDUCTORS, INC. *Kinetis Bootloader v1.2.0 Reference Manual* [online]. 2015 [cit. 2015-12-29]. Dostupné z: http://cache.freescale.com/files/soft_dev_tools/doc/ref_manual/KBTLDR120RM.pdf

[4] ŠIMEK, Václav. *Minerva - kompletní schéma zapojení* [online]. 2012 [cit. 2015-12-28]. Dostupné z: http://minerva.php5.cz/doc/schemata/Minervafull.pdf

[5] Cortex-M4 Processor. *ARM Ltd.* [online]. b.r. [cit. 2015-12-29]. Dostupné z: http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php

[6] AXELSON, Jan. *USB complete: everything you need to develop custom USB peripherals*. 3rd ed. Madison, WI: Lakeview Research, 2005. ISBN 19-314-4802-7.

[7] SNELL, Derek. *Freescale USB Mass Storage Device Bootloader* [online]. 2011 [cit. 2015-12-30]. Dostupné z: http://cache.nxp.com/files/microcontrollers/doc/app_note/AN4379.pdf

[8] DOHNAL, Michal. *Systém pro programování výukového kitu Minerva*. Brno, 2015. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bidlo Michal.

[9] LwIP: A Lightweight TCP/IP stack. *Savannah* [online]. 2015 [cit. 2015-12-30]. Dostupné z: http://savannah.nongnu.org/projects/lwip/

[10] AXELSON, Jan. *Serial Port Complete COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*. 2nd ed. Madison: Lakeview Research, 2007. ISBN 978-193-1448-079.

[11] NXP SEMICONDUCTORS. *Software and Tools for Kinetis MCUs* [online]. 2016 [cit. 2016-04-23]. Dostupné z: http://www.nxp.com/products/software-and-tools/run-time-software/kinetis-software-and-tools:KINETIS_SWTOOLS

[12] Cortex-M4 Devices Generic User Guide: Core Registers. *ARM Ltd.* [online]. 2010 [cit. 2016-04-22]. Dostupné z: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CHDBIBGJ.html

[13] BUCHTA, Petr. *Framework pro vývoj aplikací na platformě ARM*. Brno, 2015. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Vašíček Zdeněk.

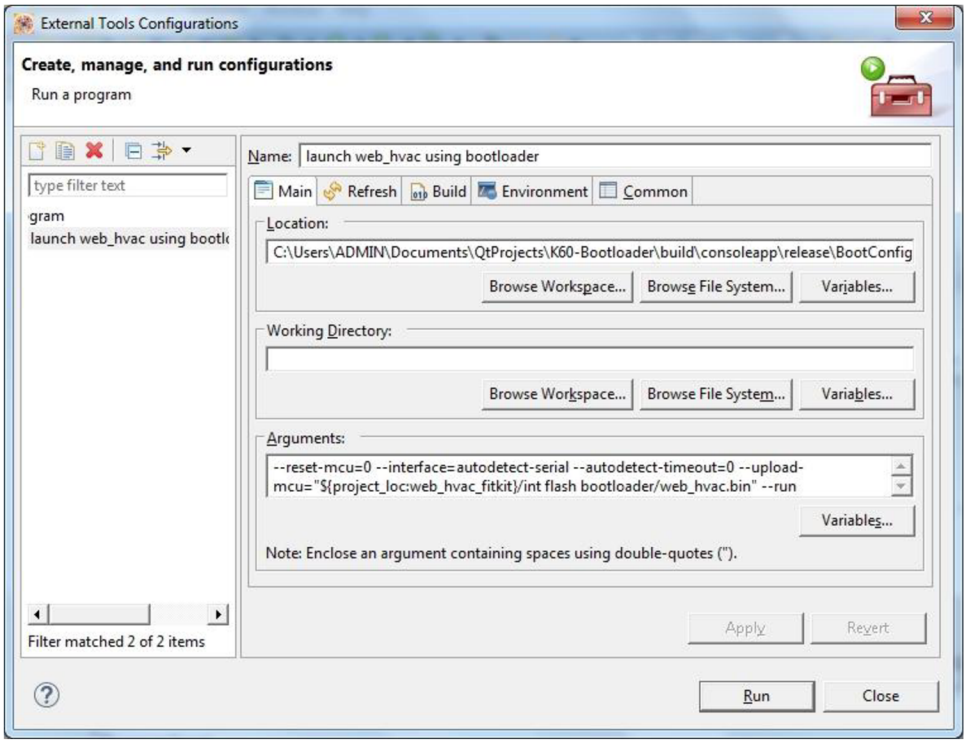
[14] FREESCALE SEMICONDUCTOR, INC. *Freescale MQX RTOS Example Quide: web_hvac example*. 2008.

Příloha A Použití zavaděče v prostředí KDS

Nastavení prostředí KDS pro nahrávání uživatelského programu pomocí zavaděče spočívá v přidání nového externího nástroje do prostředí a jeho propojení s požadovaným projektem. Nastavení se provádí pomocí *Menu → Run → External Tools → External Tools Configurations…* (viz obrázek A.1). V otevřeném okně je potřeba pod položkou *Program* vytvořit novou konfiguraci kliknutím na ikonu *„New Launch Configuration"*. V konfiguraci se dále nastaví cesta ke spustitelnému souboru konzolové aplikace a do pole Arguments se vepíší argumenty předávané konzolové aplikaci, například:

--reset-mcu=0 --interface=autodetect-serial --autodetect-timeout=0 --upload-mcu= "${project_loc:web_hvac_fitkit}/int flash bootloader/web_hvac.bin" --run

Při předávání cesty ke spustitelnému souboru lze také využívat vnitřních proměnných prostředí KDS. V tomto případě byla použita proměnná *$project_loc*, která vrací absolutní cestu k adresáři zvoleného projektu. Dále je také vhodné zrušit volbu *„Launch in background"* na záložce *Common*. Nahrávání uživatelského programu díky tomu bude probíhat na popředí stejně, jako tomu je při nahrávání programu přes rozhraní OSBDM. Ostatní nastavení mohou zůstat na výchozích hodnotách.



A.1 Konfigurace zavedení programu pomocí konzolové aplikace v prostředí KDS

Příloha B Diagram tříd aplikační knihovny

Příloha C Komunikační protokol

C.1 Typ zprávy

Typ zprávy	Datová část	Popis
0x00	Ne	Ukončení komunikace se zavaděčem. Není-li po ukončení připojeno žádné jiné rozhraní, způsobí buď přechod do stavu WAITING nebo spuštění nahrané aplikace.
0x01	Ne	Zahájení komunikace. Pokud se zavaděč nachází ve stavu WAITING, způsobí zpráva přechod do stavu CONNECTED.
0x44	Ano	Zpráva nese příkaz pro jádro zavaděče nebo odpověď na příkaz od zavaděče. Záleží na směru přenosu zprávy (od/do zavaděče).

C.2 Podporované příkazy

Kód příkazu	Název	Popis
0x00	UPLOAD	Příkaz pro zápis části souboru s uživatelským firmware do interní flash paměti. Data příkazu: - Datová část souboru (max. 2 KB) nebo žádná data pro indikaci konce souboru. Data odpovědi: - Žádné
0x01	DOWNLOAD	Příkaz pro vyčtení části uživatelského firmware z interní flash paměti. Data příkazu: - Žádné Data odpovědi: - Datová část souboru (max. 2 KB) nebo žádná data pro indikaci konce souboru.
0x02	RUN	Příkaz na spuštění nahraného firmware. Data příkazu: - Žádné Data odpovědi: - Žádné
0x03	ABORT	Příkaz na přerušení aktuální transakce. Data příkazu: - Žádné Data odpovědi: - Žádné

0x04	SET_PARAM	Příkaz pro nastavení parametrů zavaděče. Data příkazu: - UINT8 – Číslo parametru - Data parametru variabilní délky Data odpovědi: - Žádné
0x05	GET_PARAM	Příkaz pro získání parametrů zavaděče. Data příkazu: - UINT8 – Číslo parametru Data odpovědi: - UINT8 – Číslo parametru - Data parametru variabilní délky
0x06	SAVE_CFG	Příkaz pro uložení stávající konfigurace do vnitřní flash paměti mikrokontroléru. Data příkazu: - Žádné Data odpovědi: - Žádné
0x80	CONNECTED	Příkaz pro zjištění výsledku detekce správné konfigurace v FPGA. Data příkazu: - Žádné Data odpovědi: - UINT8 – 1 = OK, 0 = neplatná konfigurace
0x81	EE_PAGE_READ	Příkaz pro čtení stránky konfigurační flash paměti FPGA Data příkazu: - UINT16 – Číslo stránky Data odpovědi: - 256 B - Obsah požadované stránky
0x82	EE_PAGE_PROG	Příkaz pro zápis stránku konfigurační flash paměti FPGA. Data příkazu: - UINT16 – Číslo stránky - 256 B – Nový obsah stránky Data odpovědi: - Žádné
0x83	EE_SECTOR_ERASE	Příkaz pro vymazání sektoru konfigurační flash paměti FPGA. Data příkazu: - UINT8 – Číslo sektoru Data odpovědi: - Žádné
0x84	EE_BULK_ERASE	Příkaz pro úplné vymazání konfigurační flash paměti FPGA. Data příkazu: - Žádné Data odpovědi: - Žádné

0x85	EE_CHECKSUM	Příkaz pro výpočet kontrolního součtu z konfigurační flash paměti FPGA. Kontrolní součet je vypočítán jako prostý bezznaménkový součet hodnot jednotlivých bajtů všech stránek modulo 2^{32}. Data příkazu: - UINT16 – Počet stránek použitých k výpočtu kontrolního součtu počínaje stránkou 0. Data odpovědi: - UINT32 – Kontrolní součet
0x86	EE_GET_STATUS	Příkaz pro vyčtení stavového slova konfigurační flash paměti FPGA. Data příkazu: - Žádné Data odpovědi: - UINT8 – Stavové slovo konfigurační flash paměti
0x87	FPGA_RESET	Příkaz pro resetování FPGA Data příkazu: - Žádné Data odpovědi: - Žádné

C.3 Podporované parametry

Číslo	Název	Příznaky	Datový typ	Popis
0x00	VERSION	R	UINT32 ○ [15 … 0] = MINOR ○ [31 … 16] = MAJOR	Číslo verze zavaděče ve formátu MAJOR.MINOR
0x01	AUTORUN	R+W	UINT8 ○ 0 = FALSE ○ 1 = TRUE	Parametr ovlivňující automatické spuštění nahraného firmware
0x02	FWSIZE	R	UINT32	Velikost nahraného firmware
0x03	FWLOADADDR	R	UINT32	Počáteční adresa uživatelského firmware ve flash paměti MCU.
0x04	AVAIL_FLAGS	R	UINT32 ○ [0] = DHCP ○ [1] = Ethernet ○ [2] = USB-CDC ○ [3] = RS-232 ○ [4] = SDHC ○ [5] = FPGA	Pole bitových příznaků identifikují dostupné funkce a rozhraní zavaděče
0x05	DHCP_USE	R+W	UINT8 ○ 0 = Statická IP ○ 1 = DHCP	Parametr ovlivňující použití služby DHCP pro získání IP adresy na rozhraní Ethernet

0x06	IPV4_IPADDR	R+W	UINT32 (Big-Endian)	Statická IPv4 adresa zavaděče ve formátu Network Byte Order.
0x07	IPV4_GATEWAY	R+W	UINT32 (Big-Endian)	Statická IPv4 adresa výchozí brány ve formátu Network Byte Order.
0x08	IPV4_SUBNET_MASK	R+W	UINT32 (Big-Endian)	Statická IPv4 maska sítě ve formátu Network Byte Order.

Příloha D Obsah CD

- Kořenový adresář
  - src_firmware – složka s projektem zavaděče pro prostředí KDS 3.1.0
  - src_software – složka s projektem pro prostředí QtCreator (aplikační knihovna, grafická a konzolová aplikace)
  - src_demo – složka s demonstračními příklady pro platformu Minerva
  - build_firmware – složka s přeloženým zavaděčem pro platformu Minerva
  - build_software – složka s přeloženým software pro OS Windows
  - build_demo – složka s přeloženým demonstračním příkladem WEB_HVAC
  - driver_usb – složka s USB CDC ovladačem pro OS Windows
  - src_text – složka se zdrojovými soubory technické zprávy
  - dp_xkrupa15.pdf – technická zpráva