České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Ondřej Semmler**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Programové vybavení převodníku USB/CAN**

Pokyny pro vypracování:

1. Pro modul převodníku USB/CAN na bázi jednočipového procesoru s integrovanými řadiči CAN a USB implementujte knihovny rozhraní CAN a USB a aplikační programové vybavení modulu pro přenos dat mezi nimi.
2. Důraz klaďte na modularitu implementace, především na snadnou přenositelnost knihoven obou rozhraní.
3. Aplikační protokol na rozhraní USB musí v maximální možné míře respektovat protokol existujícího ovladače obdobného převodníku pod operační systém Windows.

Seznam odborné literatury:

[1] Kocourek, P., Novák, J.: Přenos informace. Skripta ČVUT, Praha 2003
[2] Talíř, D.: Převodník rozhraní USB-CAN. Diplomová práce ČVUT FEL, 2006
[3] Novák, V.: Programové vybavení převodníku USB-CAN. Bakalářská práce ČVUT FEL, 2007

Vedoucí: Doc.Ing. Jiří Novák, Ph.D.

Platnost zadání: do konce letního semestru 2012/2013

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 28. 11. 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Programové vybavení převodníku USB/CAN

Bc. Ondřej Semmler

Vedoucí práce: doc. Ing. Jiří Novák, Ph.D.

Studijní program: Otevřená informatika, Magisterský

Obor: Počítačové inženýrství

10. května 2012

Poděkování

Tímto bych rád poděkoval všem, kteří mi pomáhali při vzniku této diplomové práce. Především chci poděkovat vedoucímu mé práce doc. Ing. Jiřímu Novákovi, Ph.D., za cenné rady a čas strávený při konzultacích.

Zvláštní poděkování patří Mgr. Andreji Sloupové a Šárce Maixnerové za korekturu textu této práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 10. 5. 2012 .....

Abstract

The aim of this project is the design and implementation of software gateway USB/CAN based on single chip processor with ARM7TDMI core and integrated controllers CAN and USB.

The project implements the gateway firmware, which is divided into individual modules, allowing easy portability or replacement of individual parts. The project also includes a software interface between the hardware used and the application layer of the gateway with which it allows implementing new features of the hardware.

Part of the project is a mechanism for updating firmware of the hardware through the USB. The firmware update allows users to change the functionality of hardware and extends the possibilities of its use.

Within the implementation of firmware update functionality an adjustment was made to the existing drivers and respective libraries for operating system Microsoft Windows XP.

Abstrakt

Cílem této diplomové práce je návrh a implementace programového vybavení převodníku USB/CAN na bázi jednočipového procesoru s jádrem ARM7TDMI a integrovanými řadiči CAN a USB.

Práce implementuje firmware převodníku, který je rozdělený do jednotlivých modulů, což umožňuje snadnou přenositelnost nebo výměnu jednotlivých částí systému. Diplomová práce navíc obsahuje softwarové rozhraní mezi použitým hardwarem a aplikační vrstvou převodníku, pomocí kterého je možné implementovat nové funkce hardwarového přípravku.

Součástí práce je mechanismus pro aktualizaci firmwaru hardwarového přípravku prostřednictvím sběrnice USB. Aktualizace firmwaru umožňuje jednoduchým způsobem měnit funkcionalitu hardwaru a rozšiřuje tak možnosti jeho využití.

Pro zavedení podpory aktualizace firmwaru byla v rámci diplomové práce provedena úprava existujícího ovladače a k němu příslušné knihovny operačního systému Microsoft Windows XP.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V současné době obsahuje většina moderních automobilů sběrnici CAN, která realizuje komunikaci jeho senzorů a funkčních jednotek. Sběrnice CAN je ale také poměrně rozšířena do dalších sfér průmyslu, kde se uplatňuje zejména v průmyslové automatizaci.

Při vývoji, testování nebo diagnostice systémů se sběrnicí CAN je nutné zajistit připojení na tuto sběrnici a případně kontrolovat komunikaci, která na ní probíhá. Existují diagnostické systémy, které tyto funkce umožňují, ale jejich pořizovací náklady a případné rozměry jsou někdy překážkou pro jejich masové využití.

Moje práce se zaměřila na komerčně dostupné řešení pro diagnostiku a testování sběrnice CAN pomocí modulárního převodníku USB/CAN, který komunikuje s osobním počítačem s operačním systémem Microsoft Windows XP. Výsledná práce poskytuje svou cenou a rozměry zajímavou alternativu k těmto relativně nákladným existujícím systémům.

Sběrnice USB je standardním rozhraním většiny dnešních PC. Díky tomu lze využívat například jednoduše přenositelný notebook, který společně s převodníkem a implementovaným firmwarem vytváří systém pro diagnostiku nebo testování sběrnice CAN.

Kapitola 2

Specifikace cíle diplomové práce

Cílem práce je navrhnout a implementovat modulární softwarový systém pro hardwarový přípravek na bázi jednočipového procesoru s jádrem ARM7TDMI a integrovanými řadiči CAN a USB. Rozbor využitých hardwarových částí přípravku je shrnut v kapitole 3.4 „Popis hardwarového přípravku“.

Implementovaný modulární systém má primárně poskytovat nástroj pro vytvoření vlastní aplikační vrstvy zmíněného hardwarového přípravku. Sekundárním cílem je zajištění modularity systému umožnující přenositelnost na jiný hardware nebo aktualizace některých částí tohoto systému. Detailní rozčlenění jednotlivých částí systému do modulů popisuje kapitola 3.1 „Model firmwaru“.

Součástí implementace modulů pro periferii CAN je možnost časovat odesílání jednotlivých rámců na sběrnici CAN a při příjmu rámců rozpoznávat čas, ve kterém do periferie dorazily. Systém by měl být také schopen určovat odchylku mezi časem, kdy mělo dojít k odeslání rámce, a časem, kdy byl rámec na sběrnici skutečně odeslán.

Po domluvě se zadavatelem práce byl nad rámec oficiálního zadání práce doplněn požadavek na možnost aktualizace firmwaru přípravku prostřednictvím rozhraní USB. Aktualizace firmwaru je v práci realizována pomocí mechanismu „Direct Firmware Update“, který popisuje kapitola 3.6.

Díky možnosti jednoduchým způsobem vytvářet nové aplikační vrstvy přípravku ve spojení s mechanismem pro aktualizaci firmwaru získává celý systém mocný nástroj pro jednoduchou změnu funkcionality tohoto přípravku.

Podle zadání má diplomová práce obsahovat implementaci již existujícího protokolu pro převodník USB/CAN. Definice tohoto protokolu je popsána v kapitole 3.5 „Protokol USB existujícího ovladače“.

Implementace výše zmíněného protokolu je záležitostí pouze aplikační vrstvy. V práci je pro demonstraci jednoduché změny funkcionality přípravku implementováno navíc několik ukázkových příkladů jiných aplikačních vrstev. Jeden z příkladů aplikační vrstvy popisuje kapitola 4.7.1 „Příklad implementace aplikační vrstvy“.

Pro potřeby aktualizace firmwaru přípravku bylo nezbytné upravit stávající ovladač pro operační systém Microsoft Windows XP a knihovnu pro tento ovladač, které implementuje diplomová práce [11].

Pro budoucí vývojáře aplikačních vrstev systému obsahuje tato diplomová práce podrobnou programátorskou dokumentaci, vygenerovanou aplikací Doxygen.

2.1 Existující implementace

Tato diplomová vychází z bakalářské práce „Programové vybavení převodníku USB-CAN“ [20] a z diplomové práce „Převodník rozhraní USB–CAN“ [11].

V rámci diplomové práce [11] byl navrhnut hardwarový přípravek převodníku USB/-CAN, protokol komunikace po sběrnici USB mezi zařízením a PC a byl vytvořen jednoduchý firmware, realizující samotný převodník.

Hlavním cílem diplomové práce [11] bylo vytvoření správně fungujícího hardwarového přípravku a protokolu pro přenos informací po sběrnici USB. Samotná implementace firmwaru byla druhořadým úkolem. Kód firmwaru nebyl strukturovaný a implementace realizující funkční část převodníku a implementace obsluhy hardwarových prostředků nebyly oddělené. Firmware také obsahoval drobné chyby. Všechny tyto faktory zapříčinily, že byl kód nepřehledný a zasahovat do něj bylo značně komplikované. Zadání diplomové práce také neobsahovalo požadavek na podporu aktualizace firmwaru prostřednictvím USB.

Bakalářská práce [20] vytváří programové rozhraní mezi hardwarovým přípravkem a GUI aplikací „CAN Explorer“, která umožňuje zaznamenávat provoz na sběrnici CAN a vysílat rámce na sběrnici CAN. Programové rozhraní bylo v rámci práce realizováno ovladačem hardwarového zařízení pro operační systém Microsoft Windows XP a dynamickou knihovnou tohoto ovladače.

Cílem mé diplomové práce bylo napsat nový firmware pro hardwarový přípravek, který bude odstraňovat nedostatky firmwaru práce [11]. V rámci podpory aktualizace firmwaru bylo také nutné upravit ovladač a knihovnu práce [20].

Kapitola 3

Analýza a návrh řešení

3.1 Model firmwaru

V této podkapitole je popis základní struktury firmwaru přípravku diplomové práce. Detailní popisy implementace jednotlivých částí firmwaru lze nalézt v kapitole 3.4.

Jedním z hlavních požadavků v zadání diplomové práce je modularita implementovaného firmwaru převodníku. Dále je vyžadována hardwarová přenositelnost knihoven rozhraní CAN a USB. Návrh firmwaru je tedy orientován především na splnění těchto požadavků. Jednotlivé logické části firmwaru jsou rozděleny do samostatných modulů.

Model firmwaru je definován jako třívrstvý vertikálně hierarchický systém, v němž jsou implementovány jednotlivé moduly. Model firmwaru je zobrazen na obrázku 3.1.

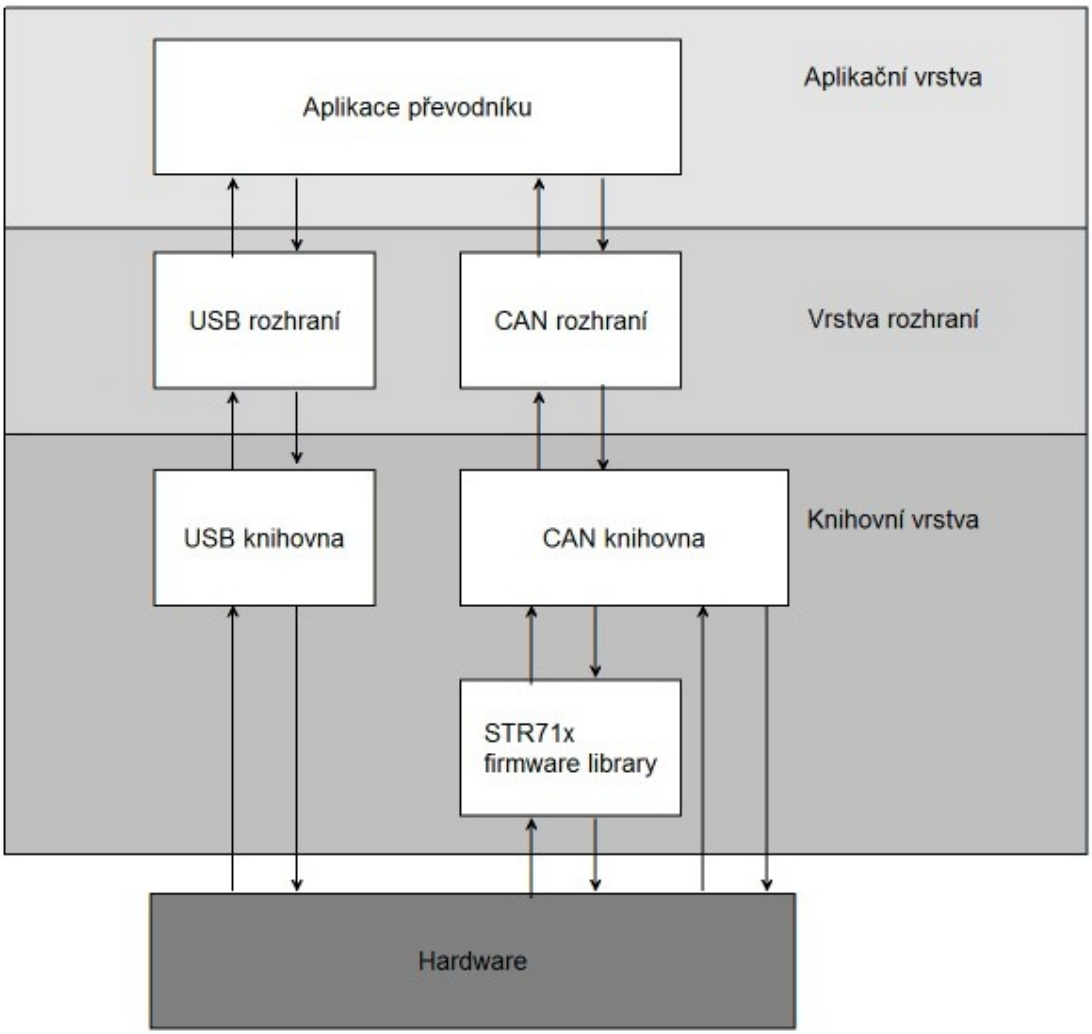
Systém definuje tři softwarové vrstvy, do kterých jsou jednotlivé moduly umístěny. Každá softwarová vrstva vykonává určitou specifickou část procesu příjmu a odesílání dat, kterou je potřeba v převodníku vykonat. Pro svou činnost využívá daná vrstva služeb své sousední nižší vrstvy a své služby dále poskytuje vrstvě vyšší.

Modul je definován jako část aplikace, jejíž jednotlivé funkce spolu logicky souvisí a společně implementují určitou separátní část aplikace, která je na ostatních modulech téměř nezávislá nebo má s jiným modulem jen slabou vazbu. Modulární systém je potom složen z jednotlivých modulů. Hlavní výhodou modulárních systémů je možnost výměny jednotlivých modulů, přičemž taková výměna má jen minimální nebo nemá žádné požadavky na úpravu modulů ostatních.

3.1.1 Knihovní vrstva

Knihovní vrstva je nejnižší vrstvou celého modelu. Její základní funkcí je přímá konektivita s hardwarovými prostředky přípravku. Knihovní vrstva obsahuje tři moduly: STR71x firmware library, USB knihovnu a CAN knihovnu.

Modul STR71x firmware library je tvořen knihovnou samotného výrobce procesoru ST Microeletronics. Tato knihovna obsahuje definice a funkce, které tvoří ovladače pro většinu periferií procesorů STR71x. Bohužel neobsahuje prostředky pro práci s periferií USB. Poskytuje však základní funkce pro práci s periferií CAN. Některé funkce této knihovny ale



Obrázek 3.1: Hierarchický model firmwaru

nejsou optimalizovány s ohledem na vysoké časové nároky vyšších vrstev. Z toho důvodu jsou některé funkce pro periferii CAN implementovány ve vyšším modulu CAN knihovně.

CAN knihovna je modul, který usnadňuje práci s periferií CAN a vytváří abstrakci nad touto periferií pro vyšší vrstvu. Jejím cílem je vytvoření minimální hardwarově nezávislé mezivrstvy, jež slouží k zajištění přímé vazby nadstavbového rozhraní na hardwaru periferie CAN. Díky tomu jsou vyšší vrstvy nezávislé na hardwaru realizujícím periferii CAN v systému. Dojde-li například k výměně hardwaru periferie CAN, lze dosáhnout opětovné funkčnosti celého systému pouhou výměnou tohoto hardwarově závislého modulu.

Modul USB knihovna je realizován knihovnou „STR7/STR9 USB developer kit“, která je detailněji popsána v kapitole 4.4. Funkce tohoto modulu je analogická modulu CAN knihovny. Implementuje práci s periferií USB na nejnižší vrstvě a vyšší vrstvě poskytuje abstraktní pohled na tuto USB periferii.

3.1.2 Vrstva rozhraní

Vrstva rozhraní poskytuje aplikační vrstvě prostředky pro práci s periferiemi CAN a USB. Tato vrstva využívá služeb hardwarově závislé knihovní vrstvy a je koncipována jako nezávislá na hardwaru periferií CAN a USB. V této vrstvě jsou implementovány funkce obecné pro převodník USB/CAN. Skládá se ze dvou modulů: USB rozhraní a CAN rozhraní.

Modul USB rozhraní poskytuje vyšší vrstvě služby pro odesílání a příjímání zpráv na sběrnici USB. Tyto služby jsou volány s parametry, které odpovídají standardu USB. Interně obsahuje funkce pro nastavení sběrnice periferie USB a enumeraci zařízení[1].

CAN rozhraní poskytuje podobně jako USB rozhraní služby pro odesílání a příjem zpráv na sběrnici CAN, nastaveni parametrů sběrnice a další. Modul vnitřně implementuje časové značky CAN zpráv, které slouží pro odesílání zpráv v požadovanou dobu. Dále je tento modul schopný simulovat více zařízení na sběrnici CAN pomocí několika samostatných front pro zprávy sběrnice CAN.

3.1.3 Aplikační vrstva

Tato vrstva je určena pro samotnou implementaci softwaru převodníku mezi sběrnicemi CAN a USB. Aplikační vrstva využívá moduly CAN rozhraní a USB rozhraní pro příjem a odesílání zpráv a nastavování parametrů jednotlivých sběrnic.

V aplikační vrstvě jsou implementovány funkce protokolů jednotlivých sběrnic a přenos dat mezi nimi. Požadavek na změnu chování hardwaru (například změna protokolu jedné ze sběrnic) lze realizovat novou implementací této vrstvy.

Přítomnost aplikační vrstvy společně se systémem DFU (kapitola 3.6) umožňuje jednoduchým způsobem změnit chování přípravku. Systém tedy může po implementaci aplikační vrstvy realizovat téměř libovolnou funkci (například simulovat jednotky automobilu).

3.2 Universal Serial Bus (USB)

Tato kapitola má za úkol ve stručnosti popsat základní vlastnosti a chování sběrnice USB. Popis nezachází do detailů a snaží se spíše seznámit budoucího vývojáře nových verzí jednotlivých modulů nebo aplikační vrstvy se základní myšlenkou fungování této sběrnice. Úplný popis sběrnice USB je obsažen ve standardu sběrnice USB [3].

3.2.1 Základní popis USB

USB je univerzální sériová sběrnice využívaná v široké škále aplikací. Standard USB definuje propojovací kabely, konektory, komunikační protokol mezi zařízeními a schopnost napájení zařízení připojených ke sběrnici.

V dnešní době se sběrnice USB používá především k propojení externích zařízení s PC (počítačová myš, klávesnice, tiskárna, kamera atd.). USB se těší velké popularitě jak mezi

[1] Enumerace zařízení je popsána v podkapitole 3.2.6.

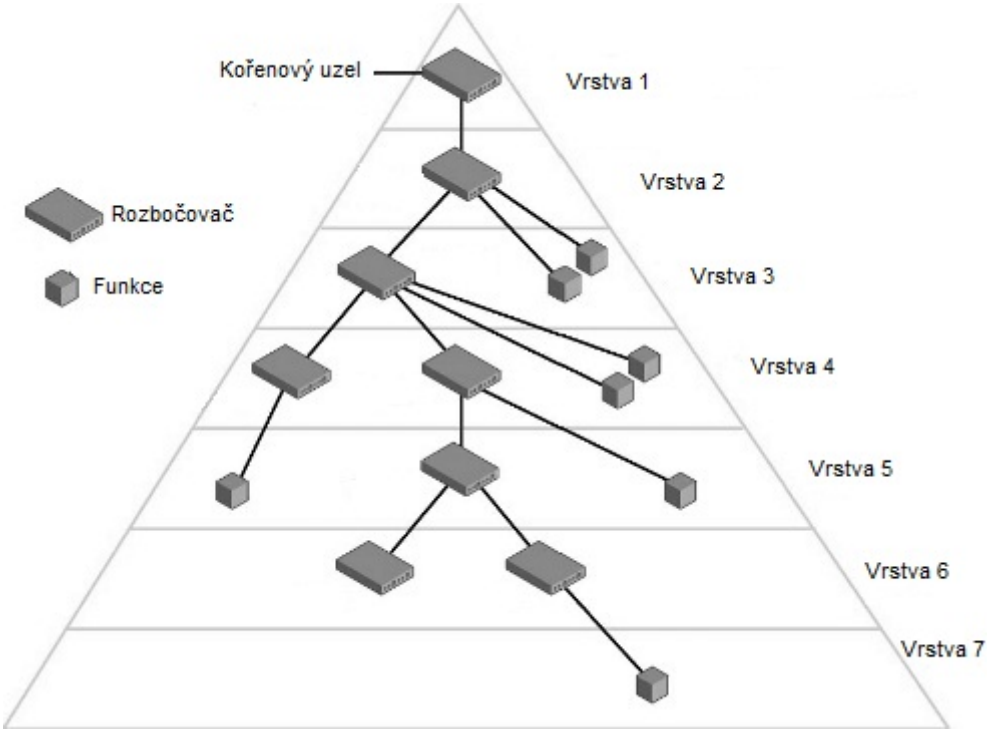
uživateli, tak i výrobci PC, proto dnes v drtivé většině PC nalezneme minimálně jeden USB port.

Sběrnice USB je sériovou sběrnicí, která propojuje jednotlivé účastníky komunikace systémem point to point. Sběrnice tedy není tvořena společnými vodiči, ale vodiče přímo propojují komunikující zařízení. Na logické úrovni se jedná skutečně o sběrnici, protože mezi komunikujícími zařízeními může docházet k větvení.

Další nespornou výhodou univerzální sériové sběrnice USB je schopnost napájet připojená zařízení, která mají dostatečně malé výkonové požadavky přes vodiče vedené kabely USB z PC. U velkého množství těchto zařízení tak odpadá nutnost napájet tato zařízení externě nebo pomocí baterií.

3.2.2 Topologie sběrnice USB

USB je sběrnice se stromovou strukturou typu master/slave. Master/slave je model komunikace, kdy jedno zařízení (typu master) řídí průběh veškeré komunikace buď mezi ním a ostatními zařízeními (typu slave), nebo mezi ostatními zařízeními vzájemně. U sběrnice USB je zařízením typu master zpravidla PC a ostatní připojená zařízení jsou typu slave.



Obrázek 3.2: Topologie sběrnice USB - zdroj [9]

Podle standardu USB [3] může být na jedné USB sběrnici až 127 dalších zařízení typu slave. Těmto zařízením se v USB terminologii říká funkce. Na obrázku 3.2 vidíme stromovou topologii USB sběrnice s připojenými funkcemi. Na vrcholu topologie se nachází takzvaný kořenový uzel. Kořenový uzel je reprezentován master USB řadičem. Tento řadič je zpravidla připojen na kořenový rozbočovač.

Rozbočovač[2] je zařízení, které slouží k větvení sběrnice USB. Samotný rozbočovač může být reprezentován specializovaným zařízením nebo zařízením, které má i jinou funkci (například rozbočovače na klávesnicích). Rozbočovač má vždy jeden konektor směřující v topologii na obrázku 3.2 nahoru (tyto konektory jsou označovány anglickým upstream port), který je připojen přímo do kořenového uzlu, nebo do dalšího rozbočovače. Většina dnešních rozbočovačů má čtyři konektory směrem dolů (downstream port), i když počet těchto konektorů není prakticky omezen.

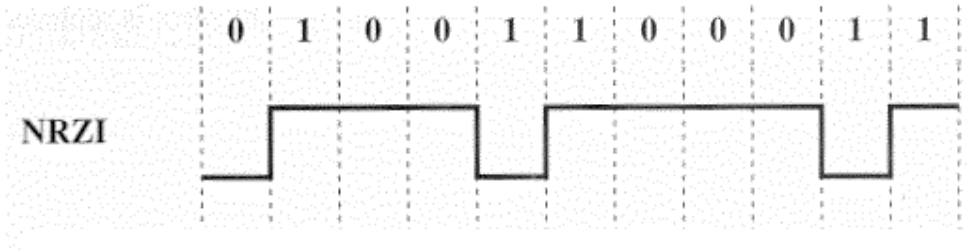
Jelikož může být na rozbočovač sběrnice USB v konečném důsledku připojeno až 127 USB zařízení, musí rozbočovat zaručit směrování paketů putujících z upstream portu směrem k nim. Tato vlastnost je zajištěna odlišením jednotlivých portů hubu a vnitřní vyrovnávací pamětí pro zasílané zprávy. Rozbočovač také zesiluje signály na sběrnici a upravuje jejich hrany. USB norma stanovuje maximální počet sériově zapojených rozbočovačů na pět. Další podstatnou vlastností všech rozbočovačů je podpora napájení připojených zařízení. Podle typu napájení můžeme rozbočovače rozdělit na rozbočovače napájené externě a na rozbočovače napájené po sběrnici USB.

3.2.3 Přenos zpráv po sběrnici USB

Přenos dat po USB sběrnici se uskutečňuje v rámcích. Rámec je časový úsek jedné milisekundy, během kterého jsou po sběrnici vysílány pakety. V jednom rámci mohou být zaslány pakety pro různá zařízení. V takovém případě dochází ke směrování paketů USB rozbočovačem.

USB paket je základní jednotkou přenosu informace na sběrnici USB. Pakety se skládají z bajtů. USB definuje čtyři typy paketů: token, handshake, data a special. Každý typ paketu má přiřazený identifikátor (PID), který je odesílán vždy na začátku paketu.

Každý přenos začíná vysláním paketu typu token kořenovým uzlem sběrnice. Paket typu token se skládá z PID a dvou bajtů obsahujících jedenáctibitovou adresu a pětibitový kontrolní součet. Následuje data paket nebo oznámení, že data nejsou přenášena. Data paket obsahuje PID následovaný 0-1023 bajty dat a šestnáctibitový kontrolní součet. Paket handshake indikuje, zda byla data úspěšně přenesena.



Obrázek 3.3: Kódovací metoda NRZI - zdroj [16]

Přenášená data jsou na sběrnici USB kódována metodou NRZI (Non Return to Zero Inverted), která je názorně předvedena na obrázku 3.3. Princip tohoto kódování je takový,

[2] Rozbočovač je v literatuře často označován pod anglickým názvem hub

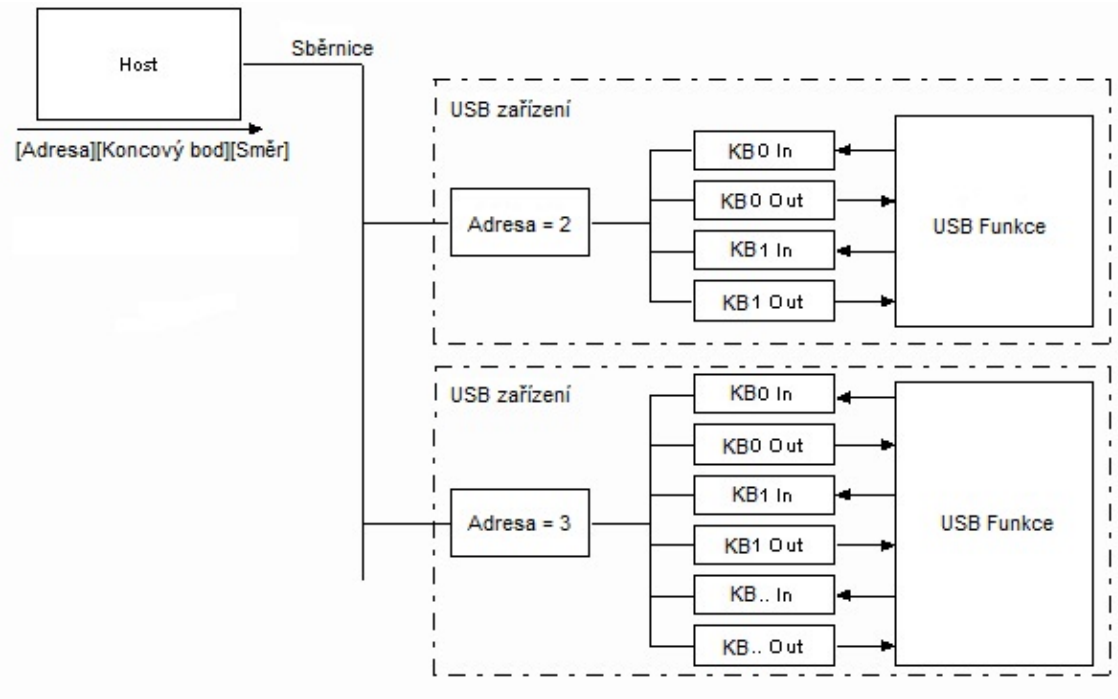
že bit logické nuly v přenášených datech vede ke změně napěťové úrovně, zatímco bit logické jedničky ponechává napěťovou úroveň beze změny.

Kromě NRZI se na přenášená data aplikuje bit-stuffing. Jedná se o vkládání bitu pro vynucení změny napěťové úrovně na sběrnici. Obsahují-li data šest po sobě jdoucích bitů logické úrovně jedna, vloží za ně vysílač automaticky jeden nulový bit. Přijímač po šesti bitech úrovně jedna očekává bit nula a automaticky tento nulový bit vložený vysílačem ignoruje.

Jak kódování metodou NRZI, tak bit-stuffing je standardní součástí hardwarového řadiče USB a programátor tedy tyto dvě techniky pro přenos dat po sběrnici USB nemusí implementovat.

3.2.4 Koncové body a typy přenosů informací

Komunikace USB zařízení je založena na logických kanálech (logical pipes). Logický kanál je spojením mezi kořenovým uzlem a koncovým bodem (endpoint) zařízení (obrázek 3.4). Každé zařízení připojené ke sběrnici USB má k dispozici až třicet dva koncových bodů: šestnáct směrem od kořenového uzlu do zařízení a šestnáct opačným směrem. Jeden koncový bod v každém směru je rezervován pro řídicí přenos sběrnice USB. Zbylých třicet koncových bodů je možné konfigurovat pro libovolný z níže uvedených typů přenosu.



Obrázek 3.4: Koncové body USB - zdroj [10]

USB standard definuje čtyři typy přenosu informací. Řídicí přenos, přerušení, isochronní přenos a blokový přenos. Každý typ přenosu je specifický a liší se různými parametry.

Řídicí přenos zaujímá až 10% času každého rámce a jako jediný přenos je obousměrný. Je využíván pro konfiguraci zařízení a zasílání různých signálů. Většina řídicích přenosů

probíhá po koncovém bodě nula[3]. Pro řídicí přenosy na koncovém bodě nula jsou definovány standardní dotazy a odpovědi, pomocí kterých je zařízení enumerováno a nastaveno.

Přerušení slouží pro spolehlivý asynchronní přenos malých dat. Typicky slouží pro zasílání přerušení do počítače. Zařízení není schopné iniciovat přenos dat na sběrnici USB. Z toho důvodu je přenos přerušení řešen pomocí periodického dotazování, zda nastalo přerušení. Perioda dotazování je nastavitelná a její hodnota je udávána v milisekundách (počet rámců).

Isochronní přenos je využíván pro přenosy, které vyžadují stálý přenos dat konstantní velikosti. Tento přenos jako jediný nemá podporu detekce chyb přenosu z důvodu preferování pravidelného datového přenosu před jeho bezchybovostí. Data přenášená tímto přenosem tedy mohou obsahovat chyby.

Pro přerušovací a pro isochronní přenos je společně rezervováno až 90% času rámce.

Blokový přenos slouží pro přenášení velkých objemů dat, které nejsou časově kritické (není potřeba data doručit do určitého času). Tento přenos využívá volného času rámce. Pokud předchozí typy přenosů využijí celý rámec, potom musí blokový přenos čekat a bude uskutečněn v některém z dalších rámců.

Typ přenosu	**Čas rámce**	**Velikost FIFO**	**Kontrola chyb**
Řídicí přenos	10%	8/16/32/64 bajtů	ano
Přerušení	90%	0-64 bajtů	ano
Isochronní přenos	90%	0-1023 bajtů	ne
Blokový přenos	0%-100%	0-64 bajtů	ano

Tabulka 3.1: Přehled typů přenosu sběrnice USB

Přehled všech typů přenosů po sběrnici USB a jejich vybrané parametry jsou shrnuty v tabulce 3.1.

3.2.5 Deskriptory USB

Každé USB zařízení obsahuje množinu deskriptorů, které poskytují kořenovému uzlu sběrnice různé informace. Deskriptory obsahují informaci o výrobci zařízení, verzi USB protokolu, kterou zařízení podporuje, počtu a konfiguraci koncových bodů zařízení a mnoho dalších informací.

[3] Každý přenos na koncovém bodě nula je řídicím přenosem. Řídicí přenosy ale mohou probíhat i na jiných koncových bodech.

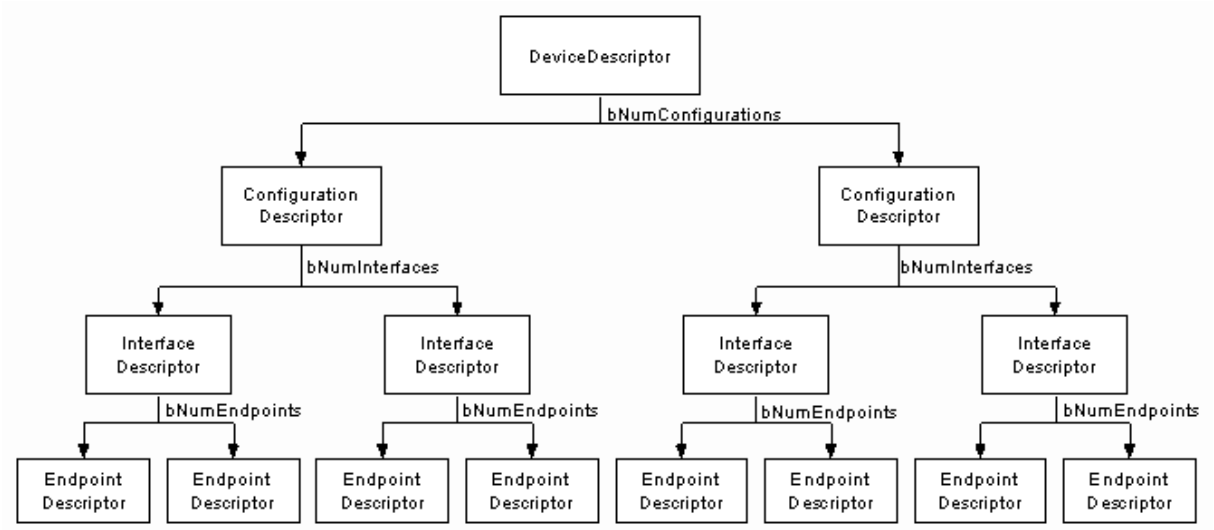
Výčet nejpoužívanějších typů deskriptorů:

- Deskriptor zařízení (Device Descriptor)
- Konfigurační deskriptor (Configuration Descriptor)
- Deskriptor rozhraní(Interface Descriptors)
- Deskriptor koncového bodu (Endpoint Descriptor)
- Textový deskriptor (String Descriptor)

Každé zařízení připojené na sběrnici USB má právě jeden deskriptor zařízení. Tento deskriptor obsahuje mimo jiné informaci o verzi USB protokolu, který zařízení podporuje, a identifikátor výrobce a produktu, podle kterých operační systém PC přiřadí k zařízení odpovídající ovladače. Dále deskriptor zařízení obsahuje informaci o počtu konfiguračních deskriptorů.

USB zařízení může pracovat v různých režimech nebo může mít více funkcí. Ke každému tomuto režimu nebo funkci je většinou přiřazen jeden konfigurační deskriptor. Ten obsahuje informaci, jakým způsobem je zařízení napájeno, hodnoty maximálního elektrického proudu, které zařízení odebírá z napájení po sběrnici USB, a počet deskriptorů rozhraní, které konkrétní konfigurace obsahuje.

Deskriptor rozhraní si lze představit jako objekt, který sjednocuje skupinu koncových bodů do celku odpovídajícího jedné funkci zařízení. Deskriptor rozhraní obsahuje informaci o počtu koncových bodů a číslo třídy a podtřídy USB zařízení. Třída USB zařízení definuje skupiny zařízení, která si jsou podobná funkcí a ke kterým má možnost operační systém přistupovat stejným nebo podobným způsobem (např. počítačová myš od různých výrobců může využívat stejný ovladač).



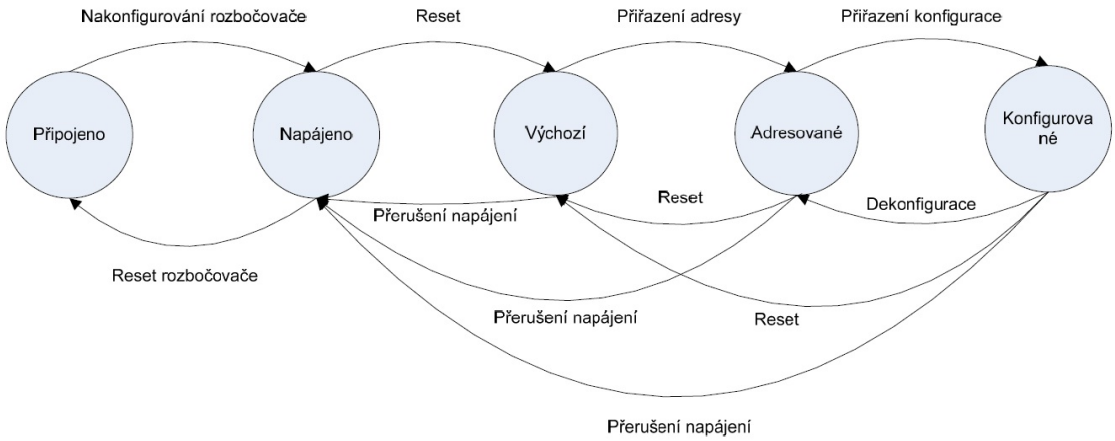
Obrázek 3.5: Hierarchie USB deskriptorů - zdroj [10]

Deskriptor koncových bodů má za úkol popisovat konfiguraci jednotlivých koncových bodů. Všechny použité koncové body, s výjimkou koncového bodu nula, musí být popsány svým deskriptorem. Ten zahrnuje adresu koncového bodu, typ přenosu, který koncový bod poskytuje (podkapitola 3.2.4), velikost maximální délky paketu, a jestliže se jedná o koncový bod poskytující přenos typu přerušení, pak i interval dotazování na přerušení kořenovým uzlem.

Textový deskriptor poskytuje textovou informaci, kterou operační systém zpravidla zobrazuje uživateli. Jedná se pouze o informativní deskriptor, který nemá vliv na funkčnost zařízení.

Obrázek 3.5 demonstruje hierarchické uspořádání deskriptorů USB zařízení.

3.2.6 Enumerace USB zařízení



Obrázek 3.6: Enumerace USB zařízení - zdroj [12]

Obrázek 3.6 znázorňuje stavový diagram enumerace zařízení připojeného ke sběrnici USB. Enumerace zařízení je proces, při kterém hostitel pomocí standardních dotazů definovaných normou vyčítá jednotlivé deskriptory (podkapitola 3.2.5) ze zařízení a na základě informací získaných z těchto deskriptorů zařízení konfiguruje a následně zavádí ovladače zařízení pro operační systém. Celý proces enumerace probíhá po koncovém bodě nula.

Zařízení vstupuje do procesu enumerace připojením do volného downstream portu rozbočovače. Rozbočovač detekuje připojené zařízení, zapne napájení portu a vyšle hostiteli informaci o připojení nového zařízení. Poté dojde k povolení portu, na který je nové zařízení připojeno, a k zaslání signálu reset do zařízení.

Po přijmutí signálu reset přejde zařízení do stavu obecného zařízení. V tomto stavu je zařízení přiřazena adresa nula. Jelikož každé zařízení na sběrnici USB musí být identifikovatelné pomocí unikátní adresy, je zřejmé, že na jedné sběrnici USB může být ve stavu obecného zařízení vždy maximálně jedno zařízení. Pokud je v jednom okamžiku detekováno připojení více zařízení, povolují rozbočovače porty těmto zařízením postupně vždy, když je ukončena enumerace předchozího zařízení.

Ve stavu obecné zařízení vyčte hostitel ze zařízení prvních osm bajtů deskriptoru zařízení. Z nich získá informaci o maximální délce paketu, který lze přenést po koncovém bodě nula. Poté hostitel přiřadí novou jedinečnou adresu v rámci sběrnice USB a zasláním signálu reset do zařízení přechází zařízení do stavu adresované zařízení.

Zařízení ve stavu adresované zařízení již komunikuje na adrese, která mu byla přidělena v minulém stavu. Nyní hostitel vyčte kompletní deskriptor zařízení a následně všechny ostatní deskriptory. Ze získaných informací hostitel vybere jednu konfiguraci zařízení a standardním příkazem nastaví tuto konfiguraci v zařízení. Tím přechází zařízení do stavu zkonfigurované zařízení a enumerace úspěšně končí.

3.3 Controller Area Network (CAN)

3.3.1 Základní popis sběrnice CAN

Controller Area Network (CAN) je sériová peer-to-peer sběrnice, která byla původně vyvinuta pro komunikaci senzorů a jednotek automobilu. Pro svou nízkou cenu, spolehlivost, jednoduchost a poměrně vysokou přenosovou rychlost se sběrnice rozšířila i do dalších průmyslových odvětví.

Peer-to-peer sběrnice je komunikační model, ve kterém jsou si všechny uzly sběrnice rovny. Na rozdíl od modelu master-slave, může být v modelu peer-to-peer iniciátorem přenosu na komunikačním médiu libovolný uzel a může řídit chování ostatních uzlů. V tomto modelu není nutné určovat, který uzel je nadřazený ostatním. Zjednodušuje se tak režie na sběrnici a zároveň se zvyšuje její spolehlivost.

CAN definuje fyzickou a logickou vrstvu OSI modelu [13]. Jednotlivé vrstvy popisují následující kapitoly.

3.3.2 Fyzická vrstva sběrnice CAN

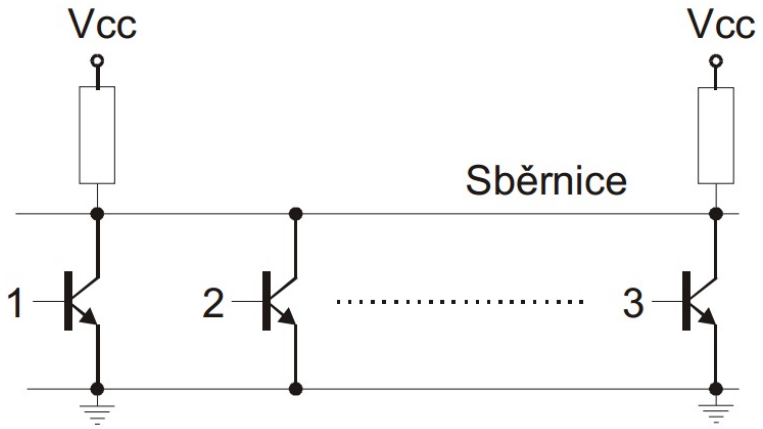
Hlavním požadavkem sběrnice CAN na fyzickou vrstvu je existence dvou stavů: dominantního stavu a recesivního stavu. Sběrnice se může v jednom okamžiku nacházet vždy v jednom z těchto dvou stavů, přičemž musí být splněna následující pravidla:

- Jestliže všechny uzly sběrnice vysílají recesivní bit, pak je sběrnice v recesivním stavu.
- Pokud alespoň jeden uzel sběrnice vysílá dominantní bit, potom se sběrnice nachází v dominantním stavu.

Recesivní a dominantní stavy tak na sběrnici realizují logické úrovně. Budeme-li považovat recesivní stav za logickou úroveň 1 a dominantní stav za logickou úroveň 0, lze předcházející pravidla realizovat pomocí operace logického součinu.

Jednoduchým příkladem realizace logického součinu je sběrnice buzená hradly s otevřeným kolektorem (obrázek 3.7).

Pro popis fyzické vrstvy sběrnice CAN je definováno několik různých norem. V těchto normách jsou definovány elektrické parametry sběrnice jako napětí a proud na sběrnici, ale



Obrázek 3.7: Logický součin realizován otevřeným kolektorem - zdroj [14]

také základní požadavky na linkovou vrstvu, jako například vícenásobný přístup na sběrnici a recesivní a dominantní stavy.

Maximální komunikační rychlost sběrnice CAN je 1 Mbit/s. Pro tuto rychlost je maximální možná délka sběrnice 40 m. S klesající přenosovou rychlostí se zvyšuje maximální možná délka sběrnice (například pro rychlost 125 kbit/s je maximální možná délka sběrnice CAN 500 m).

3.3.3 Linková vrstva sběrnice CAN

Linková vrstva sběrnice CAN se skládá z podvrstev MAC (Medium Access Control) a LLC (Logical Link Control).

Podvrstva MAC řídí přístup jednotlivých uzlů k fyzickému médiu a zajišťuje, aby jednotlivé uzly nevysílaly současně. MAC dále zajišťuje vysílání rámců sběrnice s ohledem na jejich prioritu, realizuje kanálové kódování, detekuje chyby sběrnice, reaguje na ně a zajišťuje potvrzování správně přijatých rámců sběrnice.

Podvrstva LLC slouží k filtraci přijímaných rámců na sběrnici a k hlášení o přetížení uzlů.

3.3.3.1 Přístup k fyzickému médiu sběrnice CAN

Jednotlivé uzly na sběrnici CAN jsou rovnocenné. Pokud uzel detekuje na sběrnici klidový stav, může začít vysílat. Tím získává sběrnici pro sebe a ostatní uzly mohou začít vysílat až po odeslání kompletního rámce. Výjimku tvoří chybové rámce, které může začít vysílat libovolný uzel, pokud detekuje chybu v právě přenášeném rámci.

Pokud v jeden okamžik zahájí vysílání více uzlů současně, nedojde k fyzické kolizi, neboť v důsledku existence logického součinu sběrnice zvítězí vysílání dominantního bitu nad vysíláním bitu recesivního. Každý vysílací uzel zároveň čte aktuální stav na sběrnici. Pokud vysílací uzel vysílá recesivní bit a detekuje, že sběrnice je v dominantním stavu, přestane vysílat. Tím je elegantním způsobem zaručeno, že nedojde ke kolizi při přístupu na fyzické

médium sběrnice, a zároveň přednostní vysílání rámců s vyšší prioritou. Tato metoda přístupu se nazývá CSMA/CR (Carrier Sense Multiple Access with Collision Resolution).

3.3.3.2 Zabezpečení dat a detekce chyb

Sběrnice CAN aplikuje na data současně několik mechanismů ochrany proti chybám: monitoring, CRC zabezpečení, vkládání bitů, kontrolu struktury rámců a potvrzení přijetí rámců.

Monitoring sběrnice provádí vždy vysílací uzel. V průběhu vysílání jednotlivých bitů rámce kontroluje, zda je na sběrnici úroveň, kterou sám vysílá. Pokud je v průběhu arbitráže (vysílá se *Pole řízení přístupu na sběrnici* - obrázek 3.8) na sběrnici detekován dominantní bit, přičemž uzel vysílá recesivní bit, dojde k přerušení vysílání tohoto uzlu. Jestliže v průběhu arbitráže vysílaný uzel detekuje recesivní bit, přičemž vysílá dominantní bit, nebo pokud kdekoliv mimo arbitráž detekuje jiný bit, než který vysílá, pak vyšle chybový příznak.

Na konci každého rámce je uveden patnáctibitový kontrolní součet, který je generován ze všech předcházejících bitů rámce pomocí polynomu klíče[4]: $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Pokud libovolný uzel sběrnice detekuje chybu kontrolního součtu, pak vysílá chybový příznak.

Při vysílání pěti bitů stejné úrovně jdoucích po sobě vkládá uzel do rámce jeden bit opačné úrovně. Tento mechanismus podporuje lepší synchronizaci uzlů sběrnice a zároveň slouží k detekování chyb. Jako v předchozích případech vysílá uzel při detekci chyby vkládání bitů chybový příznak.

Kontrola zpráv probíhá na základě formátů rámců v souladu se standardem CAN. Pokud uzel detekuje na některé pozici hodnotu bitu, která neodpovídá standardu, je generován chybový příznak.

Mechanismus potvrzování přijetí zprávy dává vysílači informaci o správném přijetí jím vysílané zprávy. Vysílač nastavuje bit ACK rámce na recesivní úroveň. Uzel, který úspěšně přijme tento rámec, nastaví tento bit do dominantní úrovně (využití logického součinu). Potvrzování rámců je prováděno všemi uzly sběrnice.

3.3.3.3 Chybové stavy uzlů

Každý uzel sběrnice CAN má ve svém řadiči implementovány dva chybové čítače. Jeden chybový čítač je určen pro vysílané rámce a druhý pro přijímané rámce. Je-li detekována chyba při vysílání rámce, je zvýšena hodnota vysílacího čítače a při detekovaní chyby při příjmu rámce je zvýšena hodnota přijímacího čítače. Naopak při bezchybném příjmu nebo vyslání rámce je hodnota příslušného čítače snížena. Podle hodnoty těchto čítačů se uzel sběrnice nachází v jednom z následujících stavů:

- Aktivní stav (error active) - uzel se nachází v aktivním stavu, pokud mají oba čítače hodnotu menší než 128. Pokud uzel nacházející se v tomto stavu detekuje chybu na sběrnici, vysílá aktivní chybový příznak (šest dominantních bitů). Vysláním aktivního chybového příznaku poškodí právě přenášený rámec a ostatní uzly sběrnice také detekují chybu a vyšlou chybový příznak.

[4] Způsob výpočtu CRC lze dohledat v článku [21].

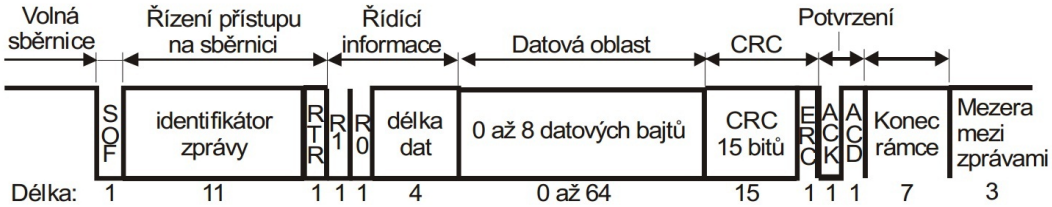
- Pasivní stav (error passive) - v pasivním stavu se nachází uzel, u kterého alespoň jeden čítač obsahuje hodnotu vyšší než 127 a nižší než 256 a zároveň vysílací chybový čítač nepřekročil hodnotu 255. Pokud uzel v pasivním stavu detekuje chybu v rámci přenášeném na sběrnici, generuje pasivní chybový rámec (šest recesivních bitů). Tento chybový rámec nepoškodí rámec právě přenášený na sběrnici.
- Odpojený stav (bus-off) - uzel je v odpojeném stavu, jestliže vysílací chybový čítač překročil hodnotu 255. Hodnota přijímacího chybové čítače nemá na přechod do odpojeného stavu žádný vliv. Uzel v tomto stavu nevysílá žádné rámce a nijak nereaguje na chyby na sběrnici. Pro ostatní uzly sběrnice se tedy jeví jako odpojený. Opuštění tohoto stavu lze dosáhnout pouze programově.

3.3.4 Zprávy sběrnice CAN

Základní jednotkou pro přenos informace po sběrnici CAN je rámec (frame). Standard CAN 2.0 definuje čtyři typy rámců: datový rámec, rámec žádosti o data, chybový rámec a rámec přetížení.

Datový rámec a rámec žádosti o data obsahují takzvaný identifikátor rámce, který určuje obsah rámce. Identifikátor není adresou příjemce ani odesílatele rámce, ale pouze identifikuje, jaká data jsou rámcem přenášena. Každý uzel sběrnice může vysílat rámce s více různými identifikátory a přijímat pouze rámce s identifikátory, o jejichž datový obsah má daný uzel zájem. Identifikátory musí být v rámci jedné sběrnice CAN jedinečné (dva uzly nesmí vysílat rámce se stejným identifikátorem).

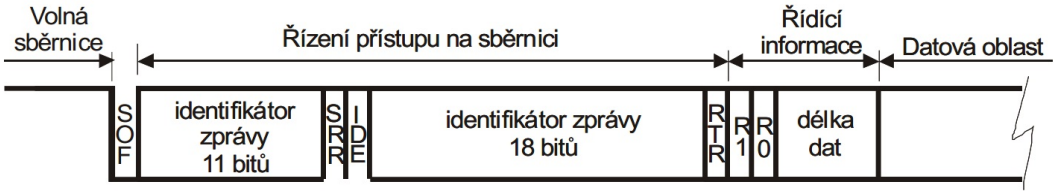
Standard CAN 2.0 se dělí na varianty A a B. Varianta A využívá jedenáctibitový identifikátor. Varianta B definuje dva typy rámců: standardní a rozšířený. Standardní rámec používá jedenáctibitový identifikátor, identifikátor rozšířeného rámce má délku dvacet devět bitů.



Obrázek 3.8: Formát datového rámce standardu CAN 2.0A - zdroj [15]

Datový rámec sběrnice CAN slouží k přenosu dat. Data v rámci mají proměnnou délku od 0 do 8 bajtů. Formát datového rámce standardu CAN 2.0A demonstruje obrázek 3.8. Rámec je rozdělen do několika polí. Vysílání začíná bitem SOF (Start Of Frame), který identifikuje začátek zprávy. Následuje pole arbitráže (řízení přístupu na sběrnici). To obsahuje jedenáctibitový identifikátor rámce a bit RTR (Remote Request), který označuje, zda se jedná o datový rámec, nebo o rámec žádosti o data. V datové zprávě musí být tento bit dominantní, v žádosti o data recesivní. Následuje pole obsahující řídící informace. To obsahuje informaci o počtu datových bajtů ve zprávě. Zpráva pokračuje datovou oblastí s jednotlivými datovými bajty, za kterými následuje jejich kontrolní součet (CRC). Zpráva je zakončena bitem potvrzení (ACK) a bitem oddělující potvrzení (ACD).

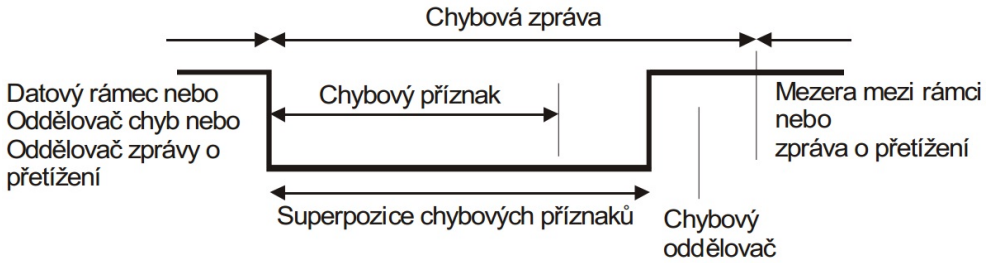
Jak již bylo uvedeno standard CAN 2.0B definuje dva typy datových rámců: standardní a rozšířený. Standardní datový rámec je převzat ze specifikace CAN 2.0A s rozdílem, že rezervovaný bit R1 je zde využit pro indikaci, zda se jedná o standardní nebo rozšířený rámec. Tento bit je označen IDE a jeho hodnota je dominantní pro standardní rámec a recesivní pro rozšířený rámec.



Obrázek 3.9: Začátek datového rámce standardu CAN 2.0B - zdroj [15]

Obrázek 3.9 ukazuje začátek rozšířeného datového rámce standardu CAN 2.0B. Delší identifikátor tohoto rámce dovoluje využívat vyšší počet rámců v systému. Na původním místě bitu RTR je bit SRR (Substitute Remote Request), který má v rozšířeném formátu vždy hodnotu recesivního bitu. Tento bit zajišťuje, že při kolizi standardního a rozšířeného rámce získá sběrnici vždy rámec standardní. Identifikátor rozšířeného rámce je rozdělen na dvě části. První část má délku 11 bitů (pro zpětnou kompatibilitu s uzly standardu CAN 2.0A) a druhá část má délku 18 bitů.

Rámec žádosti o data slouží k zaslání požadavků o data ostatním uzlům sběrnice, které jsou spojeny s konkrétním identifikátorem. Tento rámec je velice podobný datovému rámci. Bit RTR je v rámci žádosti o data nastaven do recesivní úrovně a chybí datová oblast. Pokud některý uzel vysílá žádost o data a zároveň jiný uzel vysílá tato žádaná data, je recesivním bitem RTR zaručeno, že přednost ve vysílání dostane uzel vysílající data. Uzel posílající žádost přestane vysílat data, o která chtěl žádat, přečte je na sběrnici a následně už nemusí žádost o data znovu vysílat.



Obrázek 3.10: Formát chybového rámce CAN - zdroj [15]

Chybový rámec slouží k signalizaci chyb na sběrnici CAN. Formát chybového rámce je znázorněn na obrázku 3.10. Každý uzel, který na sběrnici detekuje libovolnou chybu (porušení některého zabezpečovacího mechanismu uvedeného v kapitole), vysílá chybový příznak. Chybový příznak odpovídá šesti po sobě jdoucím dominantním nebo recesivním bitům v závislosti na aktuálním chybovém stavu uzlu, který jej vysílá. Při generování recesivního

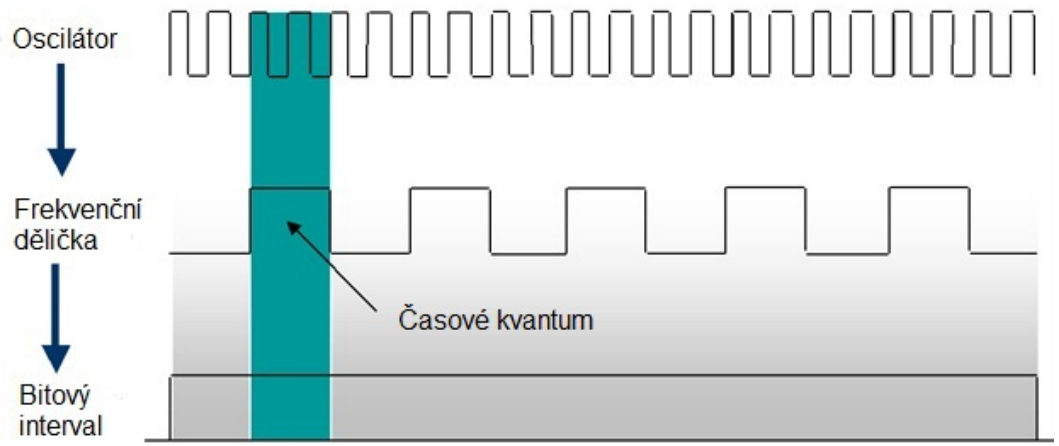
chybového příznaku nedochází k poškození vysílaného rámce. Naopak při vyslání dominantních bitů v chybovém příznaku dochází k poškození zprávy, a proto i ostatní uzly začnou vysílat chybové příznaky. Celkový rámec chyby je potom superpozicí těchto jednotlivých chybových příznaků. Po odvysílání chybového rámce následuje oddělovač chybových rámců tvořený recesivními bity.

Obrázek 3.11: Formát přetěžovacího rámce CAN - zdroj [15]

Rámec přetížení vysílají uzly, které nejsou vzhledem ke svému vytížení schopny přijímat a dále zpracovávat nové rámce. Vyslání tohoto rámce oddálí vyslání dalších rámců. Formát rámce je podobný rámci signalizace chyb. Struktura rámce přetížení je uvedena na obrázku 3.11. Rámec přetížení je složen z příznaku přetížení, který je reprezentován šesti dominantními bity a sedmi recesivními bity, které tvoří oddělovač rámce přetížení.

3.3.5 Časování sběrnice CAN

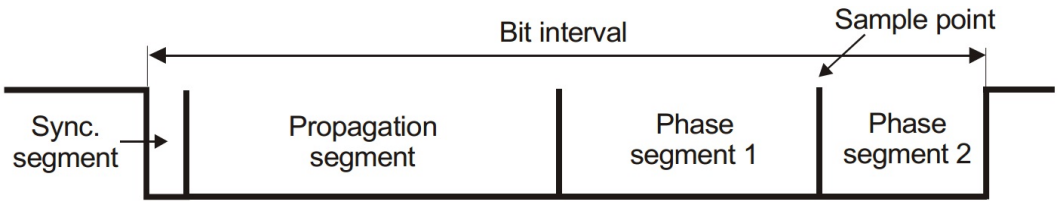
Všechny uzly jedné sběrnice CAN mají nastavenou stejnou nominální přenosovou rychlost. V důsledku nedokonalosti oscilátorů jednotlivých uzlů se ale skutečné rychlosti mírně liší. Dalším problémem, který nastává na sběrnici CAN, je doba šíření informace mezi jednotlivými uzly. Doba šíření je zatížena zpožděním na vysílači, vedení a přijímači uzlu. Z těchto důvodů je potřeba kompenzovat statické zpoždění a průběžně synchronizovat uzly.



Obrázek 3.12: Vztah oscilátoru, časového kvanta a bitového intervalu - zdroj [6]

Základní jednotkou při časování sběrnice CAN je tzv. časové kvantum (time quantum). Jak je patrné z obrázku 3.12, časové kvantum je doba, která odpovídá celému násobku frekvence oscilátoru. Z celistvého počtu časových kvant se poté skládá tzv. bitový interval.

Bitový interval je doba, za kterou se na sběrnici CAN přenese jeden bit. Bitový interval se skládá z 8 až 25 časových kvant a je rozdělen do čtyř segmentů. Strukturu bitového intervalu ilustruje obrázek 3.13.



Obrázek 3.13: Struktura bitového intervalu - zdroj [15]

Prvním segmentem je synchronizační segment, který se skládá z jednoho časového kvanta. Pokud dochází ke změně stavu sběrnice, tato změna se očekává právě v synchronizačním segmentu. Propagation segment slouží ke kompenzaci statického zpoždění mezi jednotlivými uzly. Phase segment 1 a 2 společně určují konkrétní čas v bitovém intervalu, ve kterém uzel vzorkuje stav sběrnice (čte recesivní nebo dominantní úroveň).

Na začátku každého rámce dochází k tzv. tvrdé synchronizaci. Sběrnice je vzorkována s periodou časového kvanta. Je-li detekován přechod z recesivní do dominantní úrovně, je dané časové kvantum považováno za synchronizační segment.

V průběhu přenášení jednotlivých bitů rámce se provádí průběžná resynchronizace pouze při přechodu sběrnice z recesivní do dominantní úrovně. Spočívá v dynamických změnách délek Phase segmentů tak, aby detekované hrany spadaly do synchronizačního segmentu. Časový rozdíl mezi očekávaným a skutečným výskytem hrany se nazývá fázovou chybou.

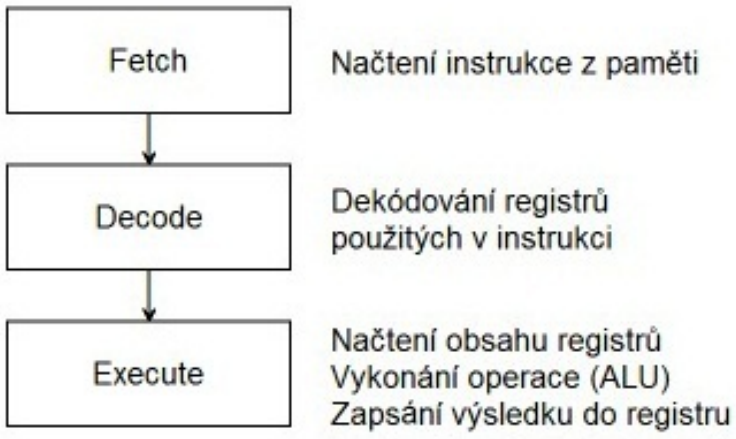
3.4 Popis hardwarového přípravku

Firmware diplomové práce je implementován na přípravku s mikrokontrolérem výrobce STMicroelectronics STR710-FZ2. Mikrokontrolér STR710-FZ2 je založen na jádře ARM7TDMI. Následující kapitoly popisují charakteristiky těchto komponent.

3.4.1 Jádro ARM7TDMI

Jádro ARM7TDMI je součástí rodiny 32-bitových mikroprocesorů ARM, která poskytuje vysoký výkon při malé spotřebě elektrické energie a malých rozměrech.

Architektura mikroprocesorů ARM je založena na redukované instrukční sadě (RISC). Mikroprocesory s redukovanou instrukční pracují s omezenou a vysoce optimalizovanou sadou jednoduchých instrukcí. Instrukce u této architektury mikroprocesorů jsou stejně dlouhé a jsou vykonávány v jednom cyklu. Mikroprocesory architektury RISC mají díky této vlastnosti vysokou propustnost instrukcí a rychlou odezvu při výskytu přerušení.



Obrázek 3.14: Pipeline jádra ARM7TDMI - zdroj [18]

Pro zrychlení vykonávání instrukcí využívá jádro ARM7TDMI třístupňové zřetězení instrukcí (pipeline). Zřetězení instrukcí zajišťuje vykonávání více instrukcí paralelně jejich rozdělením instrukcí do tří stupňů, přičemž každý ze stupňů je vykonáván jinou částí pipeline. Na obrázku 3.14 jsou znázorněny konkrétní stupně pipeline jádra ARM7TDMI.

Paměť jádra ARM7TDMI má Von Neumannovu architekturu, pro niž je charakteristická společná paměť pro data a instrukce.

3.4.2 Blokové schéma

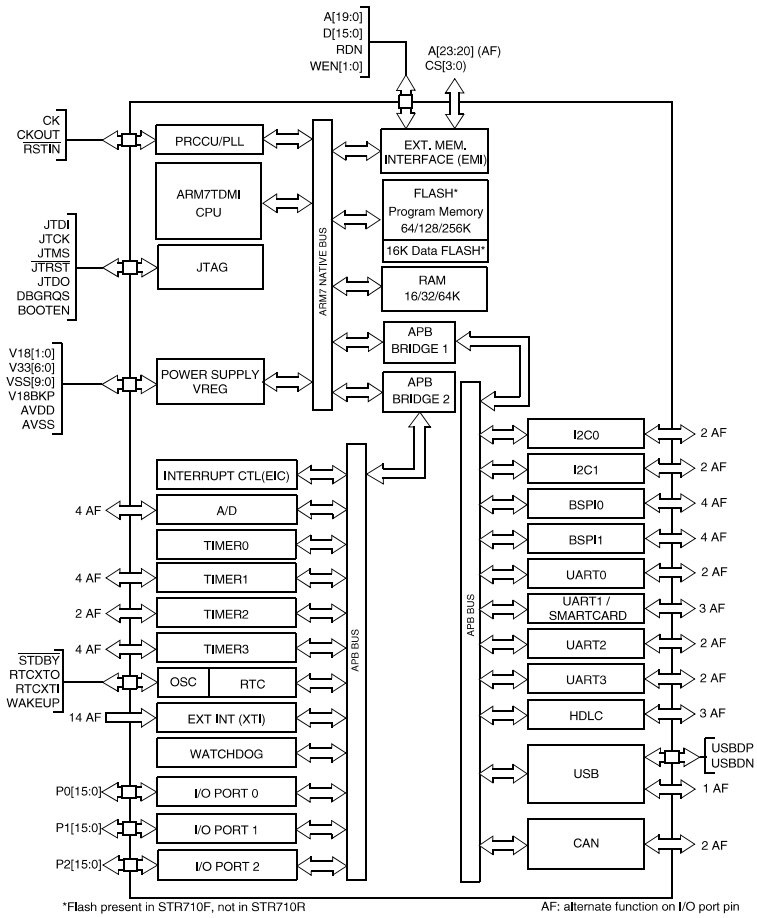
Blokové schéma mikrokontroléru na obrázku 3.15 znázorňuje rozložení jednotlivých rozhraní a periferií. V horní části schématu se nachází nativní sběrnice ARM7. Ta spojuje jádro ARM7TDMI, RAM, flash paměť, rozhraní externí paměti (EMI), jednotku řídící napájení a jednotku řídící hodinový signál mikrokontroléru (PRCCU).

Pomocí můstků APB sběrnice jsou k nativní sběrnici připojeny dvě APB (Advanced Periphetal Bus) sběrnice. APB sběrnice slouží pro připojení dalších periferií a rozhraní. První sběrnice APB1 obsahuje sériové periferie. Druhá sběrnice APB2 obsahuje systémové periferie a řadič přerušení.

Sběrnice APB jsou řízeny hodinovým signálem s nižší frekvencí než samotné jádro ARM7 a mají celkově jednodušší architekturu, což přináší nižší spotřebu elektrické energie na úkor nižší datové propustnosti. Jedná se o paralelní sběrnici s komunikačním modelem Master – Slave. APB můstek představuje vždy uzel typu Slave hierarchicky nadřazené sběrnice a zároveň jediný uzel typu Master v podřazené sběrnici APB [12].

3.4.3 Organizace paměti

Bajty jsou v paměti uloženy ve formátu Little Endian. Nejméně významný bajt je v paměti uložen na nejnižší adrese a za ním následují ostatní bajty až po nejvýznamnější.

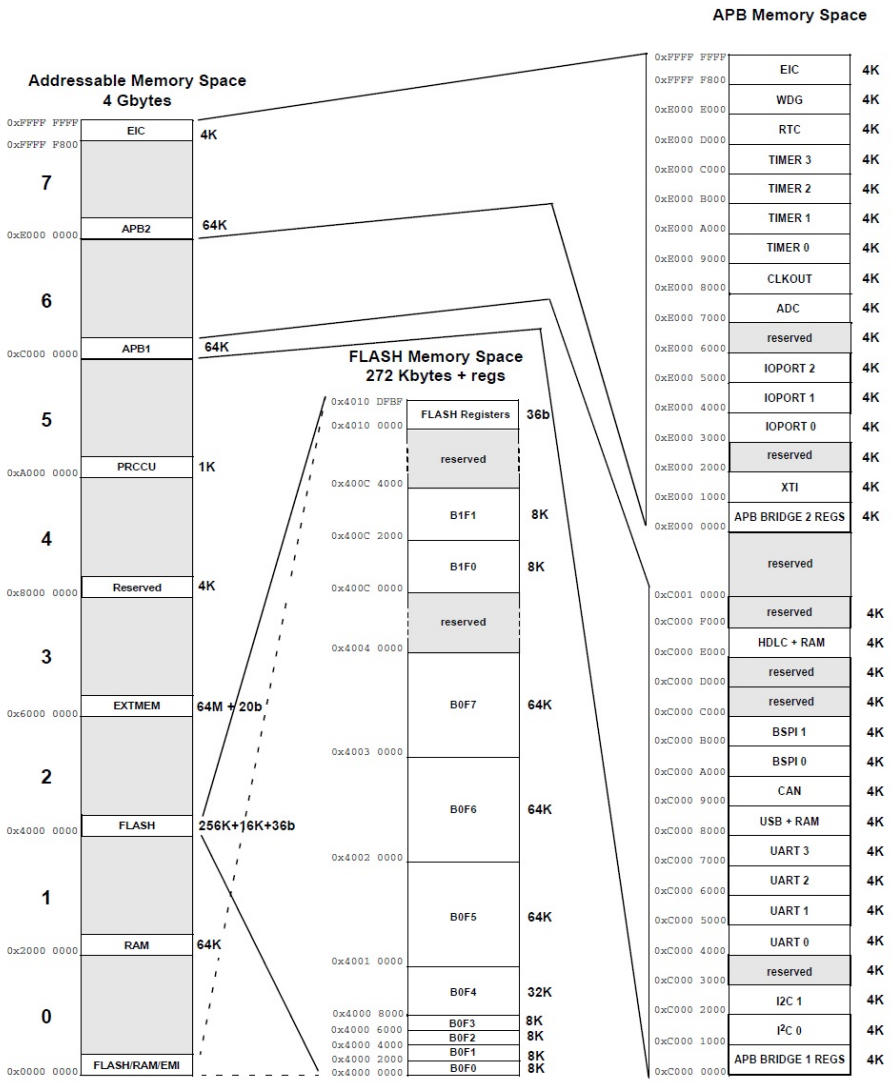


Obrázek 3.15: Blokové schéma mikrokontroléru STR710 - zdroj [18]

Bity 31-29 adresy	**Blok paměti**
000	Zaváděcí paměť
001	Paměť RAM
010	Flash paměť
011	Externí paměť
100	Rezervováno
101	Registry jednotky PRCCU
110	Sběrnice APB1
111	Sběrnice APB2

Tabulka 3.2: Přehled hlavních bloků paměti

Programová paměť, paměť dat, registry a vstupně-výstupní brány jsou organizovány

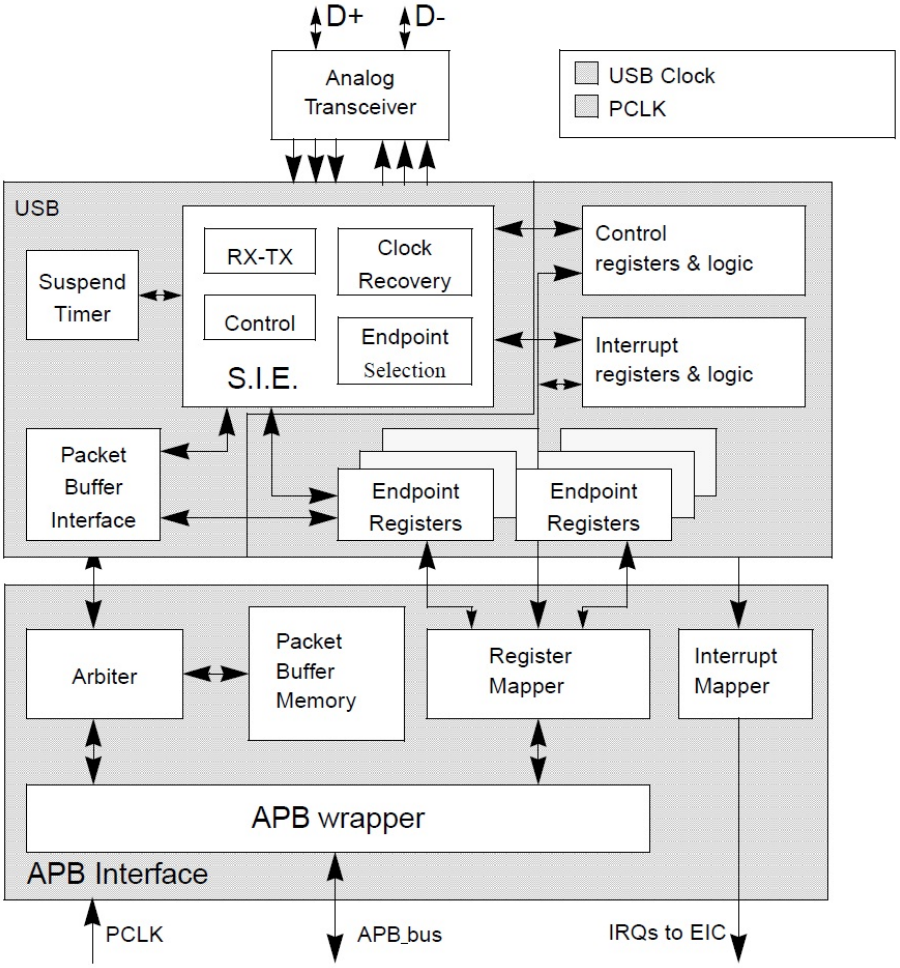


Obrázek 3.16: Mapa paměti mikrokontroléru STR710 - zdroj [18]

v jednom lineárním adresním prostoru o velikosti 4GB[5]. Na obrázku 3.16 je znázorněno mapování jednotlivých částí systému do paměti.

Procesor začíná po restartu vykonávat instrukce uložené od paměti nula. Na této paměti je v závislosti na konfiguraci mapována flash paměť, RAM nebo externí paměť. Přípravek, na kterém byla vyvíjena diplomová práce, má od paměti nula pevně konfigurováno mapování flash paměti.

Adresovatelná paměť je rozdělena do osmi hlavních bloků, které jsou identifikovány třemi nejvýznamnějšími bity adresy. Tabulka 3.2 obsahuje seznam těchto osmi bloků.



Obrázek 3.17: Blokové schéma USB periferie - zdroj [18]

3.4.4 Periferie USB

Součástí procesorů z řady STR7x je komplexní periferie USB, která implementuje rozhraní mezi full-speed sběrnicí USB 2.0 a sběrnicí APB1. Periferie poskytuje až 16 obousměrných koncových bodů, generování a kontrolu cyklického kontrolní součtu (CRC), podporu NRZI kódování a bit-stuffing, všechny standardní typy přenosů po USB a dvě přerušení.

Periferie USB poskytuje firmwaru implementovanému v mikrokontroléru spojení s hostitelským PC. Periferie detekuje a zpracovává pakety USB sběrnice, spravuje odesílaná a přijímaná data, generuje a přijímá pakety handshake podle standardu USB. Formátování zpráv, generování a kontrola CRC jsou realizovány hardwarově přímo v řadiči.

Blokové schéma USB periferie znázorňuje obrázek 3.17. Periferie je rozdělena na dvě části: USB rozhraní a APB rozhraní.

USB rozhraní periferie implementuje veškeré funkce potřebné ke komunikaci po sběrnici USB. Blok Serial Interface Engine (SIE) je jádrem rozhraní. Jeho hlavními funkcemi jsou

[5] Velikost paměťového prostoru vychází ze šířky 32-bitové sběrnice.

přenos dat, synchronizace, bit-stuffing, generování a kontrola CRC, generování a potvrzování PID a správa handshake paketů. Suspend Timer generuje hodinový signál pro synchronizaci externích zařízení a detekuje signál pro přechod do suspend režimu. Packet Buffer Interface spravuje mapování paměti pro příjem a odesílání dat. Blok Endpoint-Related Registers obsahuje registry pro práci s koncovými body. Blok Control Registers implementuje registry pro nastavení rozhraní obsahující stavové informace rozhraní a Interrupt Registers blok informuje o typu přerušení rozhraní a další doprovodné informace.

Rozhraní APB periferie USB slouží jako prostředník mezi rozhraním USB a sběrnicí APB. Packet Memory blok fyzicky obsahuje data přenášená po sběrnici USB. APB Wrapper zpřístupňuje data a registry uložené v periferii sběrnici APB a provádí mapování celé periferie USB do adresního prostoru APB.

3.4.4.1 Registry periferie USB

Periferie USB je programově řízena pomocí sady registrů, které jsou mapovány přímo do paměti mikrokontroléru. Registry lze rozdělit do dvou kategorií: obecné registry a registry pro koncové body.

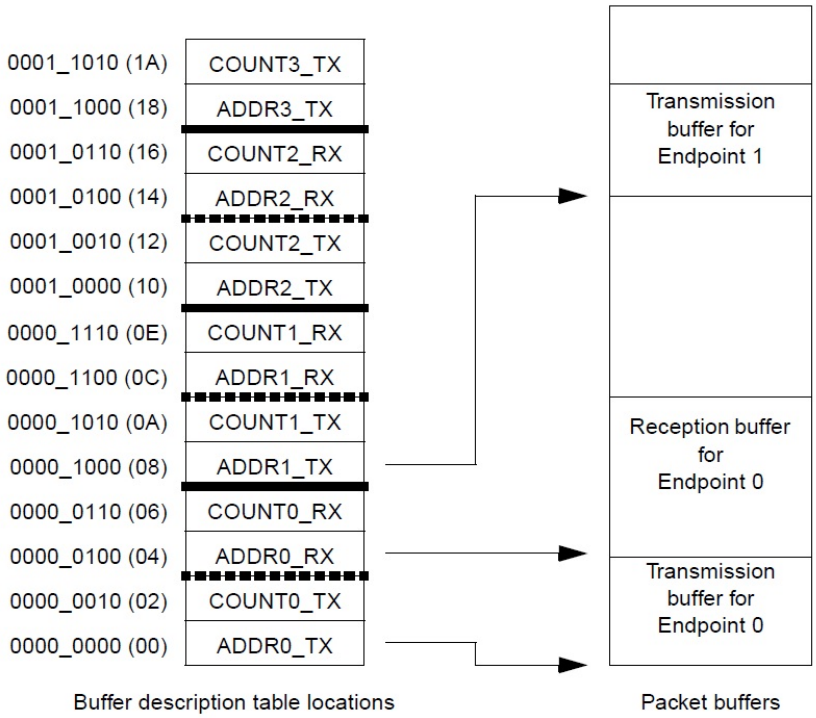
Obecné registry periferie USB definují chování periferie, její základní funkce, přerušení periferie, adresování zařízení a nesou informace o stavu periferie. Následuje výčet obecných registrů s jejich krátkým popisem.

- Control Register - definuje typy přerušení, které má periferie generovat. Řídí napájení periferie a přechod do suspend módu.
- Interrupt Status Register - udržuje informaci o přerušeních, která byla vyvolána periferií. Při přenosu dat na sběrnici obsahuje informaci o směru přenosu a koncovém bodu, který přenos realizoval.
- Frame Number Register - nese informaci o rámcích USB.
- Device Address - obsahuje adresu USB zařízení. Zároveň obsahuje bit, který povoluje periferii.
- Buffer Table Address - obsahuje adresu struktury přijímacích a odesílacích bufferů koncových bodů.

Jednotlivým koncovým bodům jsou přiřazeny řídicí registry, které zároveň informují o konkrétních stavech daných koncových bodů. Mikrokontrolér STR71x obsahuje 16 registrů pro koncové body, z nichž každý popisuje jeden koncový bod. Tento registr obsahuje informace o úspěšnosti přenosů po sběrnici a adrese, stavu a typu koncových bodů.

Data přenášená po sběrnici USB ukládá periferie do datových bufferů. Data v těchto bufferech jsou ukládána po dvoubajtových slabikách a adresována po čtyřbajtových slovech. V paměti bufferů jsou tedy ve čtyřech bajtech uloženy dva bajty USB dat a dva bajty zůstávají nevyužity. Je tedy výhodné při kopírovaní obsahu těchto bufferů kopírovat pouze dolní poloviny čtyřbajtových slov obsahujících informaci.

Každý koncový bod přenáší data samostatně. Proto je nutná existence přijímacího a odesílacího datového bufferu pro každý používaný koncový bod. Pro tento účel obsahuje periferie



Obrázek 3.18: Struktura přijímacích a odesílacích bufferů - zdroj [18]

výše zmíněný registr Buffer Table Address obsahující adresu struktury popisující přijímací a odesílací buffery koncových bodů.

Část struktury popisující datové buffery USB je zachycena na obrázku 3.18. Struktura obsahuje pro každý z koncových bodů čtyři čísla: ADDRn_TX, COUNTn_TX, ADDRn_RX a COUNTn_RX (kde n značí číslo koncového bodu). ADDRn_TX obsahuje adresu počátku odesílacího bufferu a COUNTn_TX počet platných bajtů v tomto bufferu (počet bajtů k odeslání). ADDRn_RX obsahuje počáteční adresu odesílacího bufferu a COUNTn_RX informuje o počtu přijatých bajtů a definuje počet a velikost bloků paměti vyhrazených pro tento buffer.

Periferie žádným způsobem nekontroluje, zda došlo k přetečení některého z bufferů. Z toho důvodu je důležité klást velký důraz na správné rozložení adres a rozsahů bufferů v paměti mikrokontroléru.

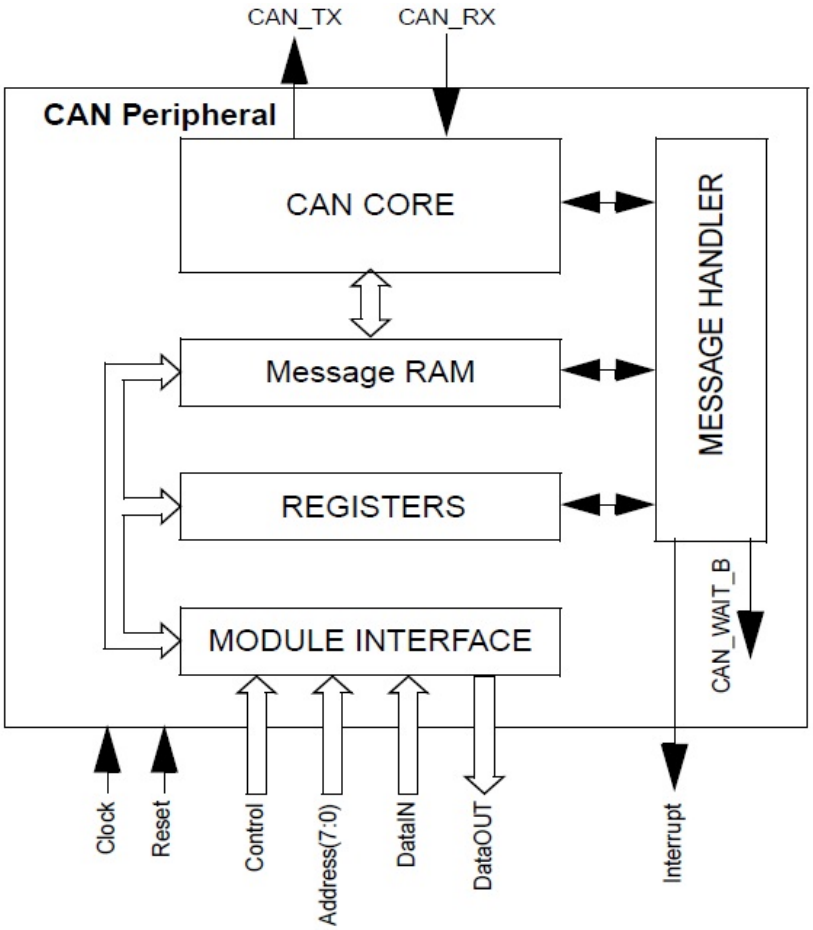
3.4.5 Periferie CAN

Procesor STR7x disponuje periferií CAN. Periferie je kompatibilní s protokolem CAN verze A a B. Podporuje tedy jak standardní formát rámce, tak i rozšířený formát rámce. Periferie je schopna komunikovat rychlostí až 1Mbit/s.

Pro komunikaci po sběrnici CAN využívá periferie tzv. objekty zpráv (Message Objects). Objekt zprávy je definován směrem komunikace (vysílání nebo příjem rámců), typem rámce

(standardní nebo rozšířený), identifikátorem rámce a maskou identifikátoru rámce[6]. Na základě těchto parametrů periferie CAN filtruje příchozí rámce a ukládá je do příslušných objektů zpráv. Při vysílání rámců se naopak plní příslušné objekty zpráv určené pro odesílání rámců.

Periferie disponuje celkem 32 objekty zpráv, které jsou uloženy v interní paměti RAM.



Obrázek 3.19: Blokový diagram periferie CAN - zdroj [18]

Obrázek 3.19 zachycuje blokovou strukturu periferie CAN. Jádrem periferie je blok *CAN CORE*. Ten implementuje protokol CAN a obsahuje posuvný registr pro sériově-paralelní konverzi zpráv posílaných a přijímaných na sběrnici CAN. Blok *Message RAM* představuje interní paměť RAM pro objekty zpráv. V bloku *REGISTERS* jsou uloženy registry pro konfiguraci a práci s periferií. *MODUL INTERFACE* je rozhraním mezi periferií a APB sběrnicí, na kterou je periferie připojena. Řadičem periferie je *MESSAGE HANDLER*. Tento blok řídí přenos informací a dat mezi jednotlivými bloky a generuje požadovaná přerušení periferie.

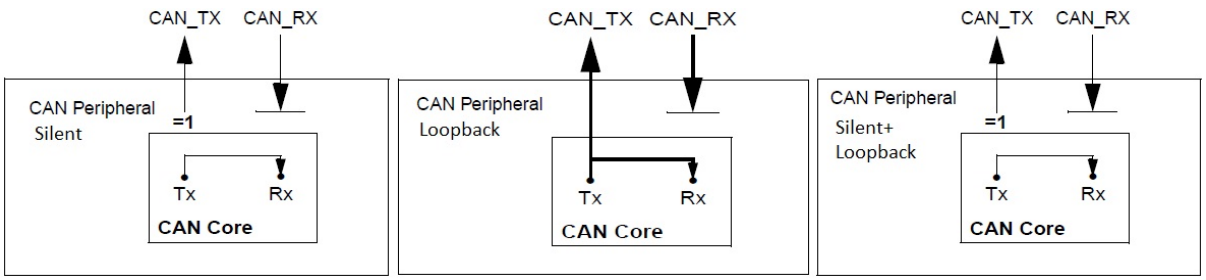
Periferie CAN disponuje funkcí automatického posílání rámců, které byly ztraceny během

[6]Identifikátor rámce a maska identifikátoru rámce společně definují množinu identifikátorů rámců akceptovaných objektem zprávy.

arbitráže sběrnice nebo se v průběhu jejich vysílání vyskytla chyba. Tato funkcionalita je volitelná a lze ji tedy vypnout správným nastavením řídicího registru.

3.4.5.1 Testovací módy periferie CAN

Periferie CAN disponuje třemi základními[7] testovacími režimy: Silent, Loopback a jejich kombinací Silent+Loopback.



Obrázek 3.20: Testovací módy periferie CAN - zdroj [18]

V režimu Silent (obrázek 3.20 vlevo) je periferie schopna pouze přijímat rámce na sběrnici CAN přičemž na sběrnici vysílá konstantně recesivní bit. Tím pádem nemůže nijak ovlivnit komunikaci na sběrnici. Tento režim lze využít pro pasivní monitoring sběrnice CAN.

Režim Loopback (obrázek 3.20 uprostřed) slouží pro testování funkčnosti periferie. V tomto režimu přijímá periferie CAN pouze vlastní odeslané rámce, které současně vysílá na sběrnici.

Posledním režimem je kombinace obou předchozích Silent+Loopback (obrázek 3.20 vpravo). Tento režim je využíván pro testování periferie bez toho, aby byla ovlivněna komunikace na sběrnici CAN.

3.4.5.2 Registry periferie CAN

Periferie CAN disponuje poměrně velkým počtem registrů. V následující kapitole budou uvedeny názvy a funkce nejdůležitějších z nich.

CAN Control Register - slouží pro povolení přerušení periferie CAN, automatického posílání chybných rámců, testovacího módu a povolení zápisu do registru obsahujícího parametry časování sběrnice CAN.

Status Register - uchovává informace o aktuálním stavu periferie. Z tohoto registru lze dále vyčíst kód LEC (Last Error Code) a informaci o úspěšném odeslání nebo příjmu rámce.

Error Counter - registr Error Counter obsahuje hodnoty chybových čítačů, jejichž funkce je detailněji popsána v kapitole 3.3.3.3 „Chybové stavy uzlů“.

Bit Timing Register - definuje časování sběrnice CAN. Pomocí tohoto registru lze nastavit periferii rychlost sběrnice CAN.

Test Register - obsahuje nastavení testovacího módu.

[7] Mezi testovací režimy není započítán režim Basic, který pracuje bez objektů zpráv.

Pro práci s objekty zpráv se využívá celá řada registrů. Pro jednoduchou práci s těmito registry je výhodné použít existující knihovnu výrobce. Případně lze jejich detailní popis dohledat v referenčním manuálu mikrokontroléru [18].

3.5 Protokol USB existujícího ovladače

Tato kapitola podrobně popisuje způsob komunikace a použitý protokol, který je využíván existujícím ovladačem pro operační systém Microsoft Windows XP. Tento ovladač byl postupně vyvíjen v pracích [12] a [11]. Implementace protokolu komunikujícího s tímto ovladačem je jedním z požadavků zadání diplomové práce.

Komunikace PC s USB zařízením probíhá prostřednictvím tří koncových bodů. Jeden koncový bod je konfigurován pro přenos typu přerušení a zbylé dva jsou konfigurovány pro blokový přenos.

3.5.1 Koncový bod typu přerušení

Koncový bod typu přerušení slouží zařízení USB k informování PC o stavu zařízení. Tento koncový bod posílá s nejkratším možným zpožděním[8] informace týkající se jak stavu celého systému, tak stavu jednotlivých periferií. Velikost dat přenášená tímto koncovým bodem je 10 bajtů a její struktura je znázorněna na obrázku 3.21.

1B	1B	1B	1B	2B		2B		1B	1B
Stavové příznaky	Nepoužito	Počet čekajících USB zpráv	Počet čekajících CAN zpráv	Počet obsazených bajtů v USB frontě		Čítač ztracených zpráv na CANu		TxErr	RxErr
				LSB	MSB	LSB	MSB		

Obrázek 3.21: Struktura dat přenášených koncovým bodem přerušení - zdroj [11]

Prvním přenášeným bajtem zprávy jsou stavové příznaky. Struktura bajtu stavových příkazů je patrná z obrázku 3.22. Nultý bit oznamuje chybu protokolu USB. Bit je nastaven, pokud zařízení obdrží příkaz, který není v protokolu definován, nebo pokud má přijatý příkaz špatnou délku nebo formát.

Bit	7	6	5	4	3	2	1	0
Význam	Plná fronta USB	Plná fronta CAN	Zařízení chce poslat data po USB	Odesílací fronta CAN není prázdná	CAN odpojený stav	CAN warning stav	CAN pasivní stav	Chyba protokolu USB

Obrázek 3.22: Význam bitů položky stavové příznaky.

Bity jedna až tři informují o stavu periferie CAN. První bit z této trojice je nastaven pokud se periferie CAN nachází v pasivním stavu, druhý pokud se nachází ve warning stavu a třetí pokud se periferie nachází v odpojeném stavu.

[8] Nejkratší možné zpoždění je 1ms, což je délka frame sběrnice USB.

Poslední čtveřice bitů je určena pro stavy interních front zařízení. Bit číslo čtyři je nastaven, pokud jsou v odesílací frontě periferie CAN rámce, které čekají na odeslání. Pátý bit je příznakem oznamujícím, že zařízení má připravena data v periferii USB pro odeslání do PC. Pokud je nastaven bit číslo šest, zařízení není schopné přijímat další rámce pro odesílání po sběrnici CAN, protože odesílací fronta této periferie je plná. Obdobně je použit bit číslo sedm pro periferii USB.

Druhý bajt v přenášené zprávě není využit.

Třetím bajtem přenášeným po koncovém bodě přerušení je počet zpráv, které jsou připraveny na odeslání do PC. Další bajt zprávy je počtem zpráv čekajících na odeslání po sběrnici CAN. Následující dvojice bajtů obsahuje informaci o počtu bajtů, které zabírají zprávy připravené k odeslání do PC, a další dvojice bajtů obsahuje číslo odpovídající počtu ztracených zpráv na sběrnici CAN. Poslední dva bajty informují o stavu chybových čítačů CAN, jejichž funkce je detailněji popsána v kapitole 3.3.3.3 „Chybové stavy uzlů“.

3.5.2 Koncové body pro blokový přenos

Existující protokol sběrnice USB definuje dva koncové body pro blokový přenos. Jeden slouží pro přenos dat z PC do zařízení a druhý pro přenos dat opačným směrem. Tato kapitola popisuje zprávy posílané po těchto koncových bodech. Zprávy směřující ze zařízení do PC jsou typu IN a zprávy směřující opačným směrem jsou typu OUT.

Následuje výčet zpráv, které jsou požity pro komunikaci mezi USB zařízením a PC. Jednotlivé zprávy jsou popsány tabulkou. V prvním sloupci je informace typu zprávy (IN nebo OUT) a poté následuje výčet jednotlivých bajtů zprávy. Pokud není v popisu jednotlivých zpráv řečeno jinak, platí, že u vícebajtových položek se posílá jako první LSB (nejméně významný bajt).

	B0	B1	B2	B3
OUT	'A'	CFG	BitRate	

	B0	B1
IN	'?'	'S'

Tabulka 3.3: Konfigurace periferie CAN.

Bit	Význam
7	Zapnout automatickou regeneraci z odpojeného stavu
6	Zapnout zpětný příjem rámců
5	Vypnout automatické opakování rámce po chybě
4	Rezervováno
3	Povolit testovací mód
2	Povolit loopback testovacího módu
1	Povolit tichý mód testovacího módu
0	Rezervováno

Tabulka 3.4: Význam bitů CFG konfigurační zprávy CAN

Zpráva **konfigurace periferie CAN** je určena ke konfiguraci různých parametrů periferie CAN. Jak demonstruje levá tabulka 3.3, je tato zpráva složena ze čtyř bajtů. Prvním bajtem zprávy je znak 'A' (hexadecimálně 0x41). Následuje bajt CFG, jehož jednotlivé bity mají význam uvedený v tabulce 3.4. Poslední dvojice bajtů definuje rychlost sběrnice CAN. Při úspěšné konfiguraci odpovídá zařízení zprávou typu IN, která je uvedena v pravé tabulce 3.3. Při neúspěchu je hlášena chyba protokolu USB.

	B0	B1	B2	B3	B4
OUT	'B'	Offset		Value	

Tabulka 3.5: Zápis do paměti periferie CAN

Tabulka 3.5 definuje zprávu pro **zápis do interní paměti periferie CAN** přípravku. Zpráva je uvedena znakem 'B', za kterým následuje dvoubajtové číslo offset určující posun od bázové adresy periferie. Zpráva je ukončena opět dvoubajtovým číslem definujícím hodnotu, která bude na offsetem určené místo zapsána.

	B0	B1	B2
OUT	'C'	Offset	

	B0	B1	B2	B3
IN	'a'	' '	Value	

Tabulka 3.6: Čtení z paměti periferie CAN

Analogickou zprávou ke zprávě zápisu do paměti periferie CAN je zpráva **čtení z paměti periferie CAN**. Zpráva pro čtení obsahuje v prvním bajt znak 'C', za kterým se nachází dvoubajtový offset adresy, která se má číst. Zařízení na tuto zprávu odpovídá čtyřmi bajty. První obsahuje znak 'a', následuje prázdný znak. Zprávu odpovědi zakončuje požadovaná dvoubajtová hodnota z paměti periferie CAN.

	B0
OUT	'D'

	B0	B1
IN	'#'	'B'

Tabulka 3.7: Obnovení periferie ze stavu odpojení

Další zprávou pracující s periferií CAN je **zpráva pro obnovení zařízení z odpojeného stavu** (tabulka 3.7). Zařízení se v odpojeném stavu žádným způsobem nepodílí na komunikaci po sběrnici CAN. Po obdržení této zprávy zařízení provede opětovnou konfiguraci periferie CAN podle poslední přijaté zprávy konfigurace periferie CAN. Detailní popis jednotlivých stavů periferie CAN je popsán v kapitole 3.3.3.3 „Chybové stavy uzlů“.

	B0
OUT	'E'

	B0	B1
IN	'*'	'C'

Tabulka 3.8: Resetování časovače

Zpráva definovaná levou tabulkou 3.8 je určena pro **vynulování interního časovače**, který slouží jako referenční čas pro odesílání a přijímání CAN rámců. Při úspěšném vynulování časovače zařízení odpovídá dvoubajtovou zprávou definovanou tabulkou vpravo.

	B0
OUT	'F'

	B0	B1
IN	'*'	'B'

Tabulka 3.9: Spuštění časovače

	B0
OUT	'G'

	B0	B1
IN	'*'	'E'

Tabulka 3.10: Zastavení časovače

Stejný formát jako zpráva pro vynulování interního časovače mají zprávy pro **spuštění a zastavení časovače**. Jediným rozdílem jsou odlišné znaky posílané těmito zprávami. Zprávy spuštění časovače definují tabulky 3.9 a zprávy zastavení časovače tabulky 3.10.

	B0
OUT	'H'

	B0	B1	B2	B3
IN	'd'	' '	Value	

Tabulka 3.11: Vyčtení hodnoty časovače

Další zprávou protokolu je zpráva pro **vyčtení hodnoty časovače** (tabulky 3.11). Dotaz na hodnotu časovače obsahuje jediný bajt s hodnotou 'H'. Odpovědí na tento dotaz je zpráva obsahující znaky 'd' a ' ', za kterými je přibližná[9] čtyřbajtová hodnota interního časovače.

	B0	B1
IN	'*'	'O'

Tabulka 3.12: Přetečení časovače

Poslední zprávou pro práci s časovačem, kterou definuje existující ovladač, je zpráva **přetečení časovače**. Tato zpráva je generována zařízením vždy, když dojde k přetečení interního časovače. Formát zprávy o přetečení časovače je uveden v tabulce 3.12.

	B0	B1	B2-B5	B6	B7	B8-B15
OUT	'I'	FLG	Timestamp	11-bit ID		Data

Tabulka 3.13: Poslání standardního rámce CAN

[9] Interní časovač běží s periodou 1μs. Délka přenosu dotazu do zařízení a zpět po sběrnici USB může trvat i několik ms. Z tohoto důvodu nelze vyčíst přesnou hodnotu časovače.

Zpráva definovaná tabulkou 3.13 slouží k **poslání standardního rámce CAN**. Prvním bajtem zprávy je znak 'I'. Následuje bajt FLG. Bajty dva až pět obsahují hodnotu interního časovače, při které má dojít k odeslání tohoto rámce na sběrnici CAN. Další dvojice bajtů nese hodnotu identifikátoru rámce CAN. Zpráva je ukončena bajty definujícími datový obsah rámce. Počet těchto bajtů je definován v položce FLG, jejíž detailní popis je obsažen v následujícím odstavci.

Bit	**Význam**
7	Rámec žádosti o data
6	Rezervováno
5	Rozšířený rámec
4	Rámec odeslaný zařízením (zpětně přijatý)
0-3	Délka datové oblasti rámce (DLC)

Tabulka 3.14: Význam bitů položky FLG zpráv rámců CAN

Význam jednotlivých bitů položky FLG objasňuje tabulka 3.14. Nejnižší čtyři bity FLG určují délku dat v bajtech, které jsou přenášeny na konci této zprávy. Pátý bit slouží k definici rámců, které byly odeslány a následně přijaty naším zařízením. Tento bit má význam pouze u zpráv týkajících se příchozích rámců, které jsou popsány níže. Bit číslo pět určuje, zda se jedná o rozšířený nebo standardní rámec CAN. Bit šest je nevyužit a bit sedm rozlišuje datové rámce a rámce žádosti o data. Jednotlivé typy rámců jsou detailněji popsány v kapitole 3.3.4 „Zprávy sběrnice CAN“.

	B0	B1	B2-B5	B6-B9	B10-B17
OUT	'J'	FLG	Timestamp	29-bit ID	Data

Tabulka 3.15: Poslání rozšířeného rámce CAN

Podobnou strukturu, jako má zpráva pro odesílání standardních rámců CAN, definuje existující protokol pro **odesílání rozšířených rámců**. Tato struktura je popsána tabulkou 3.15. Jediným rozdílem mezi těmito dvěma zprávami je délka položky obsahující identifikátor rámce CAN. U zprávy pro rozšířený rámec byla délka této položky prodloužena ze dvou bajtů na čtyři bajty.

	B0	B1	B2-B5	B6	B7	B8-B15
IN	'g'	FLG	Timestamp	11-bit ID		Data

Tabulka 3.16: Příjem standardního rámce CAN

Zprávy určené pro **příjem standardního a rozšířeného rámce CAN** jsou definovány tabulkami 3.16 a 3.17. Význam jednotlivých položek těchto zpráv je shodný s významem položek zpráv sloužících pro odesílání rámců CAN. Jediným rozdílem je typ těchto zpráv (zprávy typu IN) a hodnota prvního bajtu zpráv.

	B0	B1	B2-B5	B6-B9	B10-B17
IN	'h'	FLG	Timestamp	29-bit ID	Data

Tabulka 3.17: Příjem rozšířeného rámce

	B0	B1	B2-B5
IN	'&'	' '	#CANFrames

Tabulka 3.18: Počet ztracených rámců CAN

Pokud jsou fronty pro posílání rámců na sběrnici CAN plné a zařízení obdrží požadavek o zařazení nového rámce do této fronty, dochází ke ztrátě tohoto rámce. **Informace o počtu ztracených rámců** je obsažena ve zprávě definované tabulkou 3.18. Po odeslání zprávy o počtu ztracených rámců se interní počítadlo vynuluje.

	B0	B1	B2-B5
IN	'i'	LEC	Timestamp

Tabulka 3.19: Warning stav zařízení CAN

LEC	**Význam**
7	Rezervováno
6	Chyba kontrolního součtu CRC
5	Zařízení poslalo dominantní bit a na sběrnici detekovalo recesivní bit (mimo arbitráž)
4	Zařízení poslalo recesivní bit a na sběrnici detekovalo dominantní bit (mimo arbitráž)
3	Žádné zařízení nepotvrdilo příjem vyslaného rámce
2	Chyba formátu rámce CAN
1	Chyba bit-stuffingu: na sběrnici detekováno více než pět bitů stejné úrovně za sebou
0	Žádná chyba nenastala

Tabulka 3.20: Význam hodnot LEC pro použitou periferii CAN

V kapitole 3.4.5 popisující periferii CAN je zmínka o **warning stavu** této periferie. Informaci, že se zařízení ocitlo v tomto stavu, posílá zařízení prostřednictvím zprávy definované tabulkou 3.19. Položka LEC (Last Error Code) obsahuje číselnou hodnotu kódu popisující poslední chybu, která byla detekována na sběrnici CAN. Tabulka 3.20 obsahující hodnoty LEC a jim odpovídající chybu je vztažena přímo na použitou periferii CAN. Zprávu uzavírá dvoubajtová hodnota interního čítače, která byla odebrána v okamžiku detekování stavu.

Podobně jako předchozí zpráva, tak i **zprávy o pasivním a odpojeném stavu** zařízení informují o přechodu periferie CAN zařízení do těchto stavů. Zpráva o pasivním stavu je popsána tabulkou 3.21 a zpráva o odpojeném stavu tabulkou 3.22.

	B0	B1	B2-B5
IN	'j'	LEC	Timestamp

Tabulka 3.21: Pasivní stav periferie CAN

	B0	B1	B2-B5
IN	'k'	LEC	Timestamp

Tabulka 3.22: Odpojené zařízení CAN

3.6 Direct Firmware Update (DFU)

Direct Firmware Update (dále pouze ve zkratce DFU) je mechanismus, který umožňuje uživateli USB zařízení aktualizovat jeho firmware. Výhodou mechanismu DFU je, že nevyžaduje žádný další hardware, neboť využívá rozhraní USB. Aktualizaci firmwaru lze pak realizovat prostřednictvím jednoduché aplikace běžící na PC.

USB zařízení podporující DFU disponuje dvěma sadami USB deskriptorů. Jedna množina deskriptorů popisuje standardní funkci USB zařízení a druhá množina deskriptorů slouží pro identifikaci DFU režimu zařízení. Změna módu zařízení mezi standardní funkcí a DFU není automatická a musí být vyvolána vnějším podnětem. Vnější podnět vyvolává buď uživatel (např. tlačítkem), nebo USB host (operační systém, resp. ovladač zařízení).

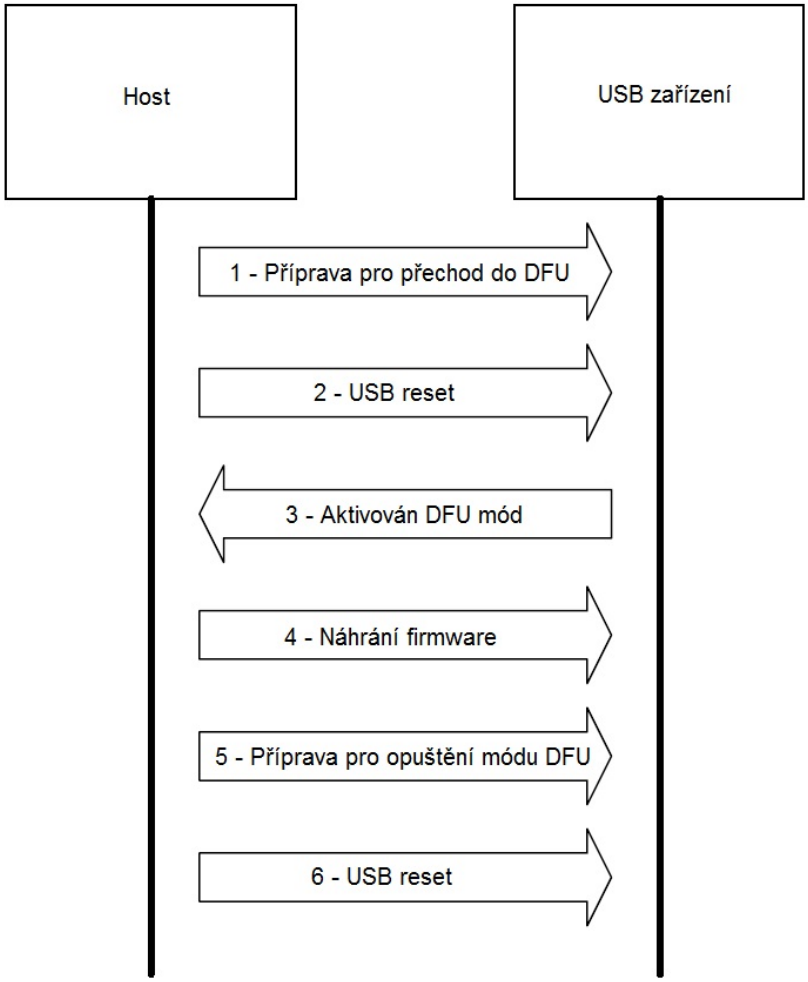
Dvě sady deskriptorů jsou zapotřebí z důvodu nahrání vhodného ovladače pro oba režimy zařízení. Běžné operační systémy nahrávají ovladače pro USB zařízení na základě ID výrobce a ID produktu, které jsou obsaženy v deskriptoru zařízení. Každé USB zařízení má právě jeden deskriptor zařízení a nelze tedy jednomu zařízení přiřadit různé ovladače (např. při změně jejich USB konfigurace). Více o deskriptorech USB pojednává kapitola „Deskriptory USB“ 3.2.5.

Zjednodušený pohled na DFU z hlediska uživatele ilustruje obrázek 3.23. Prvním krokem je příprava na přechod do režimu DFU. Tento krok má za následek, že při následném restartu zařízení bude zařízení v režimu podporující DFU. Druhým krokem je zmíněný restart zařízení. V dalším kroku se USB zařízení již enumeruje pomocí sady deskriptorů učených pro DFU. Následuje krok, ve kterém dochází k samotnému přenosu firmwaru. Poté přichází požadavek na přepnutí zařízení do standardního režimu a po restartu zařízení je již spuštěn nově nahraný firmware.

3.6.1 Komunikace DFU

Přenos dat a příkazů mechanismu DFU probíhá po koncovém bodě nula. Pro tento koncový bod definuje standard USB základní množinu příkazů sloužící především pro enumeraci a konfiguraci zařízení.

Standard DFU verze 1.1 přidává ke standardním příkazům koncového bodu nula příkazy pro řízení a přenos dat režimu DFU. Detailní popis přidaných příkazů je obsažen v tabulce 3.23.



Obrázek 3.23: Typický průběh mechanismu DFU

Význam jednotlivých příkazů DFU bude vysvětlen během popisu jednotlivých fází DFU, který obsahuje následující kapitola.

bmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**	**Data**
00100001b	DFU_DETACH	wTimeout	Interface	0	Žádná
00100001b	DFU_DNLOAD	wBlockNum	Interface	Délka	Firmware
10100001b	DFU_UPLOAD	0	Interface	Délka	Firmware
10100001b	DFU_GETSTATUS	0	Interface	6	Status
00100001b	DFU_CLRSTATUS	0	Interface	0	Žádná
10100001b	DFU_GETSTATE	0	Interface	1	Žádná
00100001b	DFU_ABORT	0	Interface	0	Žádná

Tabulka 3.23: Přehled příkazů definovaných DFU

3.6.2 Fáze DFU

Z pohledu zařízení definuje mechanismus DFU čtyři fáze: enumeraci, rekonfiguraci, přenos a manifestaci.

Následující kapitoly rámcově popisují jednotlivé fáze DFU. Detailní popis DFU a jeho jednotlivých fází naleznete ve specifikaci „USB Device Class Specification for DFU“ [2].

3.6.2.1 Fáze enumerace

Jak bylo již řečeno, zařízení DFU obsahuje dvě sady nezávislých deskriptorů: pro standardní režim a pro DFU režim. Ve fázi enumerace dochází k vyčtení deskriptorů standardního režimu.

Během normálního běhu USB zařízení je zařízení popsáno sadou standardních deskriptorů. Jednotlivé části standardní sady deskriptorů jsou rozděleny pomocí dvou deskriptorů rozhraní. Jeden definuje normální funkci USB a druhý informuje o podpoře DFU režimu.

Deskriptor rozhraní, který má za úkol informovat o podpoře DFU režimu, definuje třídu zařízení USB jako DFU a nedefinuje žádné koncové body (DFU režim pracuje pouze na koncovém bodě nula). Podle těchto informací rozpozná hostitelský operační systém, že se jedná o zařízení s podporou DFU.

3.6.2.2 Fáze rekonfigurace

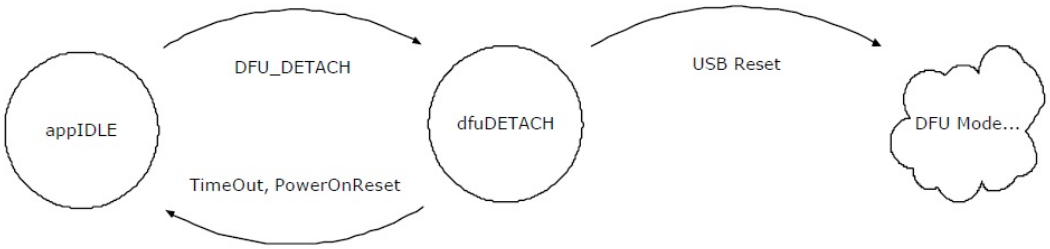
Aktualizace firmwaru zařízení je iniciována aplikací běžící na PC. Uživatel v této aplikaci definuje DFU zařízení a soubor obsahující nový firmware, který má být do zařízení přenesen.

Následuje samotná rekonfigurace zařízení. Ta se skládá ze tří operací:

- Host (PC) zašle příkaz DFU_DETACH na koncový bod nula.
- Host resetuje USB zařízení.
- Zařízení se enumeruje pomocí sady deskriptorů DFU režimu.

Pokud zařízení obdrží příkaz DFU_DETACH, přejde z běžného režimu do stavu dfuDETACH. Položka *wValue* tohoto příkazu definuje čas v milisekundách, po který zařízení čeká na příkaz USB reset na sběrnici USB. Při detekování USB resetu před uplynutím zmíněného času dojde k resetu zařízení a přechodu do režimu DFU. Pokud nedojde k detekci USB reset, zařízení se vrací do svého standardního režimu. Proces rekonfigurace znázorňuje diagram na obrázku 3.24.

Obecně lze ale říci, že způsob přechodu mezi standardním a DFU režimem zařízení není pro správné fungování DFU zásadní a vývojář firmwaru může pro tento přechod zvolit libovolný funkční mechanismus.



Obrázek 3.24: Stavový diagram průběhu rekonfigurace DFU zařízení - zdroj [2]

3.6.2.3 Fáze přenos

Přenosová fáze slouží k přenosu firmwaru z PC do zařízení (dále download), ale také k přenosu firmwaru ze zařízení do PC (dále upload).

Obrazy (soubory s firmwarem) přenášené do zařízení jsou definovány nejenom svým obsahem, ale také adresou v zařízení, na kterou se mají uložit, svojí velikostí a dalšími parametry souvisejícími s jejich strukturou a funkcí. Aplikace PC pro aktualizaci firmwaru žádným způsobem nekontroluje, zda je obraz kompatibilní se zařízením, do kterého se firmware přenáší. Proto je potřeba, aby tuto kompatibilitu zajistil výrobce zařízení, potažmo vývojář firmwaru.

Download obrazu do zařízení je realizován přenesením N paketů příkazu DFU_DNLOAD. Číslo N lze získat z rovnice N=((F−S)/O)+1, kde F je velikost obrazu, S je velikost informační části obrazu (adresa pro nahrání, délka obrazu, ID výrobce atd.) a O je optimalizovaná velikost dat pro přenos v jednom paketu koncového bodu nula. Všechny proměnné v rovnici mají rozměr bajt.

Mezi jednotlivými pakety DFU_DNLOAD se host dotazuje DFU zařízení na jeho stav prostřednictvím příkazu DFU_GETSTATUS. Stav zařízení definuje, zda došlo k úspěšnému přenesení dané části obrazu.

Po úspěšném přenesení poslední části obrazu pošle host zařízení příkaz DFU_DNLOAD s daty o nulové délce. Tím informuje o dokončení přenosu obrazu a fáze přenosu tímto příkazem končí.

Funkce **upload** je v DFU přítomna pro vyčtení a zálohování aktuálního firmwaru v zařízení. Dále může sloužit pro zpětnou kontrolu firmwaru nahraného do zařízení.

Upload je inverzní k funkci download, což umožňuje zpětný download obrazů získaných uploadem. Proces upload probíhá zasíláním příkazu DFU_UPLOAD hostem. Zařízení odpovídá pakety s částmi obrazu. Po odeslání poslední části odpoví zařízení paketem obsahujícím identifikátor konce obrazu. Zařízení je zodpovědné za správné formátování dat obrazu a výběr adresy pro upload obrazu. Po obdržení celého obrazu připojí aplikace hosta za data obrazu informační část obrazu, čímž je proces uploadu ukončen.

3.6.2.4 Fáze manifestace

Po dokončení downloadu obrazu do zařízení přechází zařízení do fáze manifestace. V této fázi dochází k samotnému nahrání firmwaru do programové paměti zařízení.

Některá zařízení provádí fázi manifestace společně s fází přenosu. To znamená, že příchozí data ihned nahrávají do příslušné programové paměti. Tento postup je nevyhnutelný, pokud je obraz vzhledem k velikosti operační paměti přípravku příliš velký.

Po úspěšné manifestaci se zařízení nachází ve stavu, kdy je opět schopno pokračovat v komunikaci DFU, nebo ve stavu, kdy čeká na USB reset a přechod do standardního režimu. Způsob ukončení manifestace závisí na nastavení v deskriptoru rozhraní DFU.

3.7 Vývojové prostředky

3.7.1 Doxygen

Pro jednoduchou a efektivní tvorbu dokumentace využívá diplomová práce Doxygen. Doxygen je multiplatformní nástroj pro generování dokumentace ze zdrojového kódu.

Doxygen podporuje generování dokumentace ze zdrojových textů programovacích jazyků C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP a C# do různých formátů (HTML, Latex, RTF, XML, PDF, PostScript a další). Generování probíhá z komentářů obsažených přímo ve zdrojovém textu. Je zapotřebí dodržovat některý ze stylů komentování, které Doxygen podporuje.

Příklad stylu komentování, který podporuje Doxygen:

```
1  /**
2  * a normal member taking two arguments and returning
3  * an integer value.
4  * @param a an integer argument.
5  * @param s a constant character pointer.
6  * @see Test()
7  * @see ~Test()
8  * @see testMeToo()
9  * @see publicVar()
10 * @return The test results
11 */
12 int testMe(int a,const char *s);
```

Doxygen je vydán pod licencí GNU General Public License. Dokumentace vygenerovaná tímto nástrojem je práce odvozená ze zdrojových kódů a není ovlivněna licencí, pod kterou je Doxygen vydáván.

3.7.2 Apache Subversion (SVN)

Při vývoji rozsáhlejších softwarových projektů je výhodné používat systémy pro verzování zdrojových kódů. Systémy pro verzování zdrojových kódů umožňují evidovat a ukládat změny ve zdrojových textech v průběhu vývoje softwaru. Takový systém vývojáři umožňuje vytvářet zálohy zdrojového kódu a při špatné úpravě kódu se vracet k předchozím verzím.

Další nespornou výhodou těchto systémů je, že umožňují spolupráci více vývojářů na stejném zdrojovém kódu. Verzovací systémy totiž hlídají a řeší případné kolize ve zdrojových

textech. Z toho důvodu se verzovací systémy staly základem pro rozvoj open source programů.

Při vývoji firmwaru diplomové práce byl použit verzovací systém Apache Subversion (SVN). SVN je vyvíjen společností CollabNet, Inc. a je šířen pod licencí umožňující jeho bezplatné komerční využití.

SVN vznikl jako náhrada staršího verzovacího systému CVS a z části je také tímto systémem inspirován. Použití SVN je ale o poznání jednodušší. Na webu výrobce [1] je SVN volně ke stažení. Tento web zároveň obsahuje dokumentaci a příklady práce s tímto systémem.

3.7.3 IAR IDE

Firmware diplomové práce byl vyvíjen pomocí integrovaného vývojového prostředí *IAR Embedded Workbench for ARM* verze 5.40. Prostředí představuje kompletní sadu nástrojů a ovladačů pro sestavování a ladění vestavěných systémů na bázi procesoru ARM.

Výrobce prostředí IAR Embedded Workbench uvádí, že kompilátor jazyka C/C++ integrovaný v prostředí je optimalizovaný pro sestavování firmwaru pro ARM procesory a generuje kompaktní a efektivní kód.

Prostředí zároveň podporuje ladění firmwaru prostřednictvím rozhraní JTAG. O ladění firmwaru pomocí tohoto rozhraní stručně pojednává kapitola 4.10 „Ladění firmware“.

3.7.4 Windows Driver Kit (WDK)

Pro úpravu ovladačů pro operační systém Microsoft Windows XP lze s výhodou použít Windows Driver Kit (WDK). WDK se skládá ze sady nástrojů, zdrojových příkladů, kompilátorů, hlavičkových souborů, knihoven a dokumentace, pomocí nichž lze vyvíjet a ladit ovladače určené pro operační systémy Microsoft Windows.

3.7.5 VMware Player a WinDbg

Pro ladění ovladače operačního systému Microsoft Windows XP byl použit software VMware Player a WinDbg.

VMware Player je volně dostupný nástroj, který vytváří virtuální stroj. Virtuální stroj vytváří virtualizované prostředí mezi platformou počítače a operačním systémem, ve kterém může koncový uživatel provozovat software na abstraktním stroji [4]. VMware Player vytváří nativní hardwarový virtuální stroj, čímž umožňuje sdílet hardwarové prostředky mezi hostitelským operačním systémem a virtualizovaným operačním systémem. Tato vlastnost je nezbytná pro ladění ovladače pro hardwarový přípravek.

Debugger WinDbg je víceúčelová počítačová GUI aplikace určená pro ladění softwaru operačních systémů Microsoft Windows. WinDbg umožňuje ladění aplikací v uživatelském režimu, ovladačů a samotného jádra operačního systému.

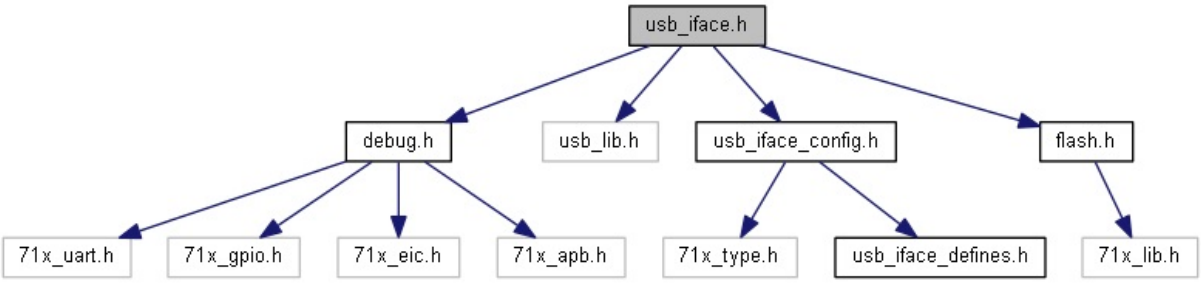
WinDbg je již součástí nejnovějších verzí WDK. Společně s nástrojem VMware Player vytvářejí soubor aplikací pro pohodlné ladění ovladačů operačního systému Microsoft Windows. Postup při vývoji ovladače pomocí této sady nástrojů je popsán v práci [12] v kapitole „Ladění ovladačů“.

Kapitola 4

Realizace

4.1 Členění souborů

Pro splnění požadavku jednoduché přenositelnosti částí firmwaru je implementace systému rozdělena do jednotlivých modulů, které mezi sebou mají minimální nebo žádnou vazbu. Struktura systému z hlediska modularity je znázorněna na obrázku 3.1 v kapitole 3.1 „Model firmwaru“.



Obrázek 4.1: Hierarchie souborů modulů USB

usb_iface.h	Hlavičkový soubor modulu
usb_iface.c	Zdrojový soubor modulu
usb_iface_config.h*	Hlavičkový soubor konfigurace modulu
usb_iface_config.c*	Zdrojový soubor konfigurace modulu
usb_iface_defines.h	Definice typů pro modul

Tabulka 4.1: Soubory modulu aplikačního rozhraní USB

Moduly lze rozdělit na dvě skupiny podle periferie, ke které přísluší: moduly USB a moduly CAN. Každý modul se skládá z jednoho nebo více souborů. V následujících odstavcích jsou popsány jednotlivé soubory modulů a jejich základní funkce.

Systém modulů pro periferii USB tvoří dva moduly: Aplikační rozhraní USB a USB knihovna. Aplikační rozhraní USB poskytuje soubor služeb pro aplikační vrstvu a USB knihovna poskytuje soubor služeb pro aplikační rozhraní.

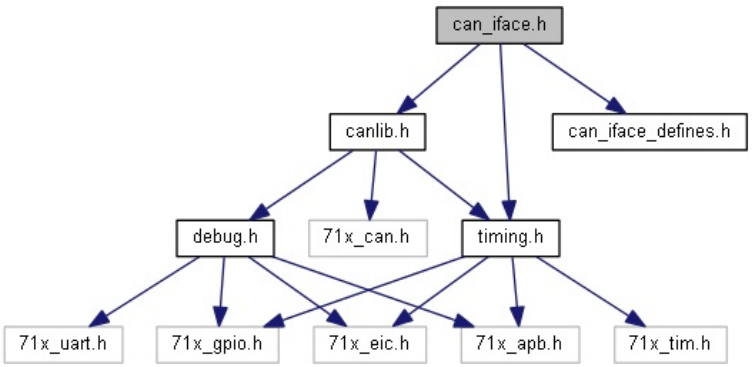
Aplikační rozhraní USB je tvořeno pěti soubory. Jejich názvy a popis shrnuje tabulka 4.1. Hvězdičkou jsou označeny soubory, které mění vývojář aplikační vrstvy.

Pro implementaci USB knihovny je využita existující knihovna poskytovaná výrobcem mikroprocesoru. Soubory, které knihovnu konfigurují a začleňují do systému, jsou popsány v kapitole 4.4 „USB knihovna“.

Na obrázku 4.1 je zobrazena hierarchie hlavičkových souborů modulů USB. Soubory *debug* slouží pro ladění aplikace prostřednictvím sériového rozhraní. Jejich popis a funkce naleznete v kapitole 4.10 „Ladění firmwaru“. Soubory *flash* zprostředkovávají funkce pro zápis a čtení slova na adrese 0x40008000 paměti flash. Význam zápisu a čtení na této adrese je objasněn v kapitole 3.6 „Direct Firmware Update (DFU)“. Soubory uvedené předponou *71x_* patří základní firmwarové knihovně výrobce. Tato knihovna je popsána v kapitole 4.2 „STR71x firmware library“. A konečně hlavičkový soubor *usb_lib.h* představuje knihovnu USB.

can_iface.h	Hlavičkový soubor modulu
can_iface.c	Zdrojový soubor modulu
can_iface_defines.h	Definice typů pro modul

Tabulka 4.2: Soubory modulu aplikačního rozhraní CAN



Obrázek 4.2: Hierarchie souborů modulů CAN

Moduly pro periferii CAN jsou stejně jako u periferie USB dva: aplikační rozhraní CAN a knihovna CAN. Soubory aplikačního rozhraní CAN shrnuje tabulka 4.2. Soubory knihovny CAN jsou *canlib.h* a *canlib.c*. Pro časování rámců sběrnice CAN jsou použity funkce implementované v souborech *timing.h* a *timing.c*. Detailnější popis modulů lze nalézt v následujících kapitolách práce.

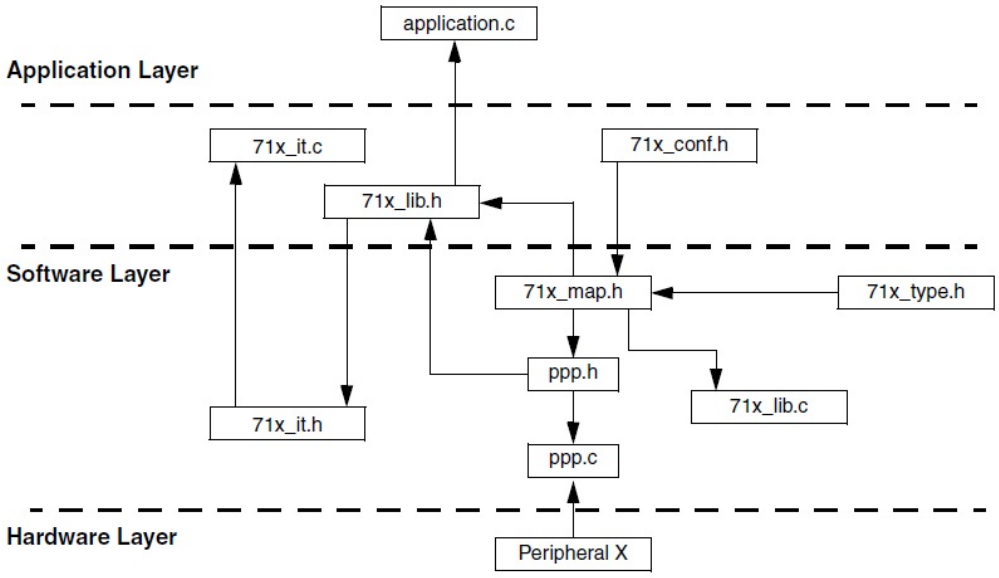
Na obrázku 4.2 je zachycena hierarchie hlavičkových souborů pro moduly CAN. Soubory *debug* a soubory uvedené předponou *71x_* jsou totožné se stejnými soubory modulů periferie USB.

4.2 STR71x firmware library

STR71x firmware library je knihovna výrobce mikroprocesoru STMicroelectronics. Knihovna obsahuje soubor firmwarových funkcí a definic, které usnadňují práci s periferiemi hardwarového přípravku. Vývojář využívající tuto knihovnu může implementovat aplikace pro přípravek bez hlubší znalosti fungování jednotlivých periferií.

Nicméně i tato knihovna může obsahovat chyby, případně některé funkce, které mohou být pro účel vývoje specifických aplikací neefektivní. Z toho důvodu je výhodné používat knihovnu obezřetně a mít při jejím využívaní alespoň základní znalosti o fungování využívaných periferií prostřednictvím přímého přístupu k jejich registrům.

Všechny registry periferií STR71x jsou mapovány do paměti. Díky tomu lze pomocí kódu programovacího jazyka C obsluhovat periferie přípravku.



Obrázek 4.3: Struktura knihovny STR71x firmware library - zdroj [17]

Obrázek 4.3 zobrazuje strukturu knihovny STR71x firmware library. Funkce a definice pro jednotlivé periferie jsou obsaženy v souborech *71x_ppp.c* a *71x_ppp.h* (kde ppp je označení konkrétní periferie). *71x_map.h* obsahuje mapování registrů periferií do paměti a definice pro všechny periferie přípravku. Hlavičkový soubor *71x_lib.h* obsahuje hlavičkové soubory všech periferií. Soubor *71x_conf.h* definuje periferie, které bude využívat aplikační vrstva. Soubory *71x_it.h* a *71x_it.c* slouží pro obsluhu přerušení přípravku.

Vývojáři aplikace stačí modifikovat soubory *71x_conf.h* a *71x_it.c* pro začlenění knihovny do jeho projektu.

Knihovna STR71x firmware library neobsahuje podporu pro periferii USB.

4.3 Aplikační rozhraní USB

Modul aplikační rozhraní USB poskytuje aplikační vrstvě systému prostředky pro manipulaci s periferií USB. Funkce a definice obsažené v aplikačním rozhraní periferie USB se snaží prezentovat tuto periferii z obecného pohledu. Umožňují tím vývojáři aplikační vrstvy vytvářet aplikace využívající tuto periferii bez nutnosti znalosti práce s USB periferií na nižší vrstvě.

Aplikační rozhraní USB je nezávislé na hardwaru USB periferie. Při změně hardwaru je potřeba pouze implementovat knihovnu USB, přičemž aplikační rozhraní zůstává zachováno.

Modul využívá služeb a funkcí, které poskytuje nižší vrstva - USB knihovna. Implementace USB knihovny je popsána v kapitole 4.4.

4.3.1 Nastavení rozhraní

Aplikační rozhraní umožňuje několika způsoby konfigurovat chování implementovaného USB zařízení.

Základním nastavením je definice ID výrobce a ID výrobku zařízení. Na základě těchto parametrů operační systém automaticky nahrává příslušný ovladač zařízení. Změna těchto parametrů umožňuje změnit ovladač, který bude se zařízením komunikovat. ID výrobce a ID výrobku jsou v aplikačním rozhraním uloženy ve struktuře, která je definována v souboru *usb_iface_config.c*. Změnu těchto parametrů lze provést jednoduchým přepsáním těchto parametrů ve zmíněném souboru.

Dalším důležitým nastavením aplikačního rozhraní USB je definování koncových bodů, které zařízení implementované aplikační vrstvou využívá. Soubor *usb_iface_config.h* obsahuje definice počtu koncových bodů typu IN a OUT. Konkrétní nastavení jednotlivých koncových bodů je implementováno v souboru *usb_iface_config.c*.

Pro předcházení problémům, které mohou nastat při záměně koncových bodů pro příjem a odesílání, implementuje aplikační rozhraní pro oba směry komunikace oddělené struktury. Při záměně koncových bodů (například posílání dat na koncový bod pro příjem) dochází k chybě při překladu.

4.3.2 Inicializace rozhraní

Prvním krokem při využívání aplikačního rozhraní USB je jeho inicializace. Ta se provádí funkcí *USB_iface_init*, která konfiguruje hodiny USB periferie, nastaví přerušení, inicializuje knihovnu USB a provádí další úkony nezbytné pro inicializaci.

V průběhu enumerace je zařízení dotazováno na jednotlivé deskriptory, které popisuje kapitola 3.2.5 „Deskriptory USB“. Ty jsou generovány aplikačním rozhraním USB pomocí šablony pro deskriptory a nastavením aplikačního rozhraní popsaného v předešlé kapitole.

4.3.3 Příjem USB dat

Pokud periferie USB přijme data na některý z koncových bodů určených pro příjem, volá nižší vrstva funkci aplikačního rozhraní, které předává číslo přijímacího koncového bodu. Funkce aplikačního rozhraní poté určí strukturu koncového bodu, který přijal data, a pomocí funkcí

nižší vrstvy vyčte data z periferie USB. Pokud je buffer aplikační vrstvy pro příchozí data volný, jsou vyčtená data uložena do něj a koncový bod je uveden do stavu, kdy může přijímat další data. Pokud je ale buffer obsazen předchozími daty, jsou data ponechána v periferii USB do doby, kdy dojde k uvolnění zmíněného přijímacího bufferu aplikační vrstvy. Pokud jsou přijatá data ponechána v periferii, je koncový bod uveden do stavu, ve kterém odmítá přijímat nová data, čímž nedojde ke ztrátě dat uložených v periferii.

Přijatá USB data uložená v přijímacím bufferu aplikačního rozhraní vyčítá aplikační vrstva prostřednictvím funkce *USB_iface_read_recv_msg*. Návratová hodnota této funkce je rovna počtu bajtů v přijímacím bufferu. Pokud jsou v přijímacím bufferu k dispozici nějaká přijatá data, jsou společně s informací o jejich velikosti a se strukturou koncového bodu pro příjem dat předána aplikační vrstvě prostřednictvím parametrů této funkce.

Aplikační vrstva může testovat, zda byla přijata data na USB, voláním funkce *USB_iface_read_recv_msg* a testováním nenulovosti její návratové hodnoty. Pokud je pro aplikační vrstvu tento způsob vyčítání neefektivní nebo je vyžadována okamžitá odezva na přijatá data, lze využít mechanismu zpětného volání (callback). Pro tento účel definuje vývojář funkci v aplikační vrstvě a pomocí funkce aplikačního rozhraní *USB_iface_register_callbacks* tuto funkci zaregistruje jako funkci pro zpětné volání. Při dalším příjmu dat provede aplikační rozhraní po uložení dat do přijímacího bufferu volání funkce zpětného volání, která byla zaregistrována.

Z důvodu zvýšení rychlosti vyčítání dat z USB nedochází mezi aplikační vrstvou a aplikačním rozhraním USB ke kopírování dat přijímacího bufferu a je předán pouze ukazatel do paměti na tato data. Po přečtení přijatých USB dat je tedy nutné, aby aplikační vrstva informovala aplikační rozhraní USB o dokončení čtení těchto dat prostřednictvím funkce *USB_iface_release_recv_msg*. Jakmile aplikační vrstva tuto informaci obdrží, uvolní přijímací buffer pro další data.

4.3.4 Odesílání USB dat

Pro odesílání zpráv prostřednictvím sběrnice USB obsahuje aplikační rozhraní USB funkci *USB_iface_send*. Parametry této funkce obsahují strukturu koncového bodu, po kterém se mají data odeslat, a samotná data k odeslání. Funkce prostřednictvím služeb poskytovaných nižší vrstvou vyhledá adresu bufferu pro zápis dat do periferie a zkopíruje data pro odeslání do tohoto bufferu. Poté nastaví příslušný koncový bod do odesílacího stavu.

Jakmile dojde k odeslání dat, je uložena informace o koncovém bodě, který odeslání uskutečnil. Pomocí funkce *USB_iface_read_sent* je poté aplikační vrstva schopna zjistit, zda došlo k odeslání dat na sběrnici USB a jaký koncový bod toto odeslání uskutečnil. Návratová hodnota funkce je jedna, pokud došlo k odeslání, a nula, pokud k odeslání nedošlo.

Podobně jako u příjmu dat poskytuje aplikační rozhraní mechanismus zpětného volání při úspěšném odeslání dat. Funkce pro zpětné volání definovaná v aplikační vrstvě se registruje funkcí *USB_iface_register_callbacks*. Aplikační rozhraní poté volá funkci zpětného volání při úspěšném odeslání dat. Tím je zaručena okamžitá odezva při odeslání dat po USB v aplikační vrstvě.

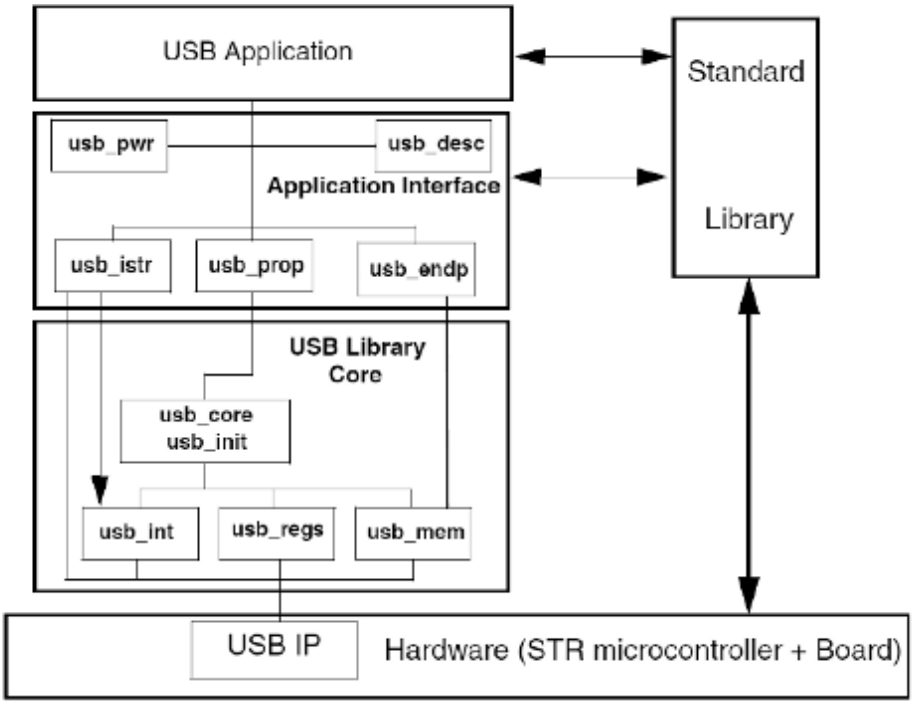
4.3.5 Další funkce

Pro informování aplikační vrstvy o připojení k PC prostřednictvím USB je aplikační rozhraní vybaveno funkcí *USB_ iface_ connected*. Návratová hodnota této funkce je rovna jedné, pokud se zařízení nachází ve zkonfigurovaném stavu. Pokud je zařízení v jakémkoliv jiném stavu, je návratová hodnota této funkce rovna nule. Jednotlivé stavy zařízení z pohledu sběrnice USB jsou popsány v kapitole 3.2.6 „Enumerace USB zařízení“.

Důležitou funkcí aplikačního rozhraní USB je funkce *USB_ iface_ enter_ DFU*. Zavoláním této funkce je vyvolán zápis slova obsahující číslo nula do paměti flash na adresu 0x400008000, což zapříčiní, že po dalším restartu přejde zařízení do režimu DFU. Zařízení v režimu DFU je připraveno aktualizovat svůj firmware. Popis funkce DFU obsahuje kapitola 3.6 „Direct Firmware Update (DFU)“. Implementace tohoto režimu je shrnuta v kapitole 4.8 „Implementace DFU“.

4.4 USB knihovna

Pro manipulaci s periferií USB je použita knihovna „STR7/STR9 USB developer kit“. Tato knihovna poskytuje kompletní softwarovou podporu pro USB periferie všech 32-bitových mikrokontrolérů výrobce STMicroelectronics a je volně ke stažení na webu zmíněného výrobce [5]. Součástí knihovny je také řada příkladů použití, které jednoduše ilustrují, jakým způsobem s knihovnou pracovat.



Obrázek 4.4: Struktura USB knihovny - zdroj [19]

Pro zabudování knihovny do diplomové práce bylo potřeba implementovat rozhraní mezi zmíněnou knihovnou a rozhraním pro aplikační vrstvu USB/CAN převodníku.

Samotná knihovna je rozdělena do dvou vrstev: *USB Library Core* a *Application Interface* (obrázek 4.4). *USB Library Core* slouží pro obsluhu a komunikaci s hardwarem a implementuje USB protokol. Knihovna je kompatibilní se standardem USB 2.0. Uživatel knihovny zanechává *USB Library Core* beze změny. Veškerá implementace programů je obsažena v *Application Interface*. Zabudování knihovny do tohoto projektu tedy vyžaduje znalost struktury *Application Interface*.

Application Interface je rozdělena do šesti skupin souborů:

- Konfigurace knihovny
- Obsluha přerušení USB
- Funkce pro přenos dat po koncových bodech
- Funkce USB zařízení
- Správa napájení
- Deskriptory USB

Soubor *usb_conf.h* implementuje **konfiguraci knihovny**. Obsahuje definici rodiny mikrokontrolerů a definici konkrétního mikrokontroléru použitého hardwarového přípravku. Dále je v konfiguraci definována bázová adresa datových bufferů pro koncové body, nastavení datových bufferů pro koncový bod nula a typy přerušení, které jsou knihovnou podporovány.

Obsluha přerušení USB je implementována v souboru *usb_istr.c*. Soubor obsahuje funkci *USB_Istr*, která je volána při přerušení periferie USB. Tato funkce rozpozná typ přerušení USB a poté volá příslušné funkce zařízení nebo funkce *USB Library Core*. Při přerušení, které vyvolal koncový bod nula, jsou volány interní funkce USB knihovny z *USB Library Core*. Uživatel má tedy ulehčenou práci a nemusí například implementovat proces enumerace USB zařízení.

Při přerušení od ostatních koncových bodů jsou volány funkce definované v souborech *usb_endp.h* a *usb_endp.c*. Tyto funkce slouží pro **přenos dat po koncových bodech**. Funkce v těchto souborech volají podle směru komunikace příslušné funkce aplikačního rozhraní USB s parametrem čísla koncového bodu. Aplikační rozhraní USB tedy dostává informaci o příjmu dat na konkrétní koncový bod nebo o úspěšném odeslání dat z daného koncového bodu.

Funkce USB zařízení implementují soubory *usb_prop.h* a *usb_prop.c*. Soubory obsahují funkce zpětného volání z *USB Library Core*. Tyto funkce jsou volány v průběhu enumerace a uživatel může definovat, jakým způsobem enumerace probíhá a co se při ní děje. Funkce *Device_Reset* je volána při detekování přerušení reset periferie USB. V této funkci dochází k inicializaci koncového bodu nula, nastavení adresy zařízení na hodnotu nula a k vygenerování obsahu struktur datových bufferů pro koncové body podle nastavení koncových bodů v aplikačním rozhraní USB (kapitola 4.3.1 „Nastavení rozhraní“). Soubory dále obsahují funkce, které předávají *USB Library Core* jednotlivé deskriptory USB, jež jsou generovány na základě šablon aplikačního rozhraní USB a nastavení koncových bodů. Ostatní funkce USB zařízení lze dohledat v dokumentaci firmwaru na přiloženém CD.

Správu napájení USB zařízení implementují soubory *usb_pwr.h* a *usb_pwr.c*.

Deskriptory USB jsou generovány aplikačním rozhraním USB, a proto nejsou definovány v žádném speciálním souboru.

Application Interface knihovny USB je implementován modifikací příkladu práce s knihovnou získaného od výrobce. Autoři knihovny sami uvádějí, že modifikací příkladů dodávaných s knihovnou lze nejlépe dosáhnout využití potenciálu této knihovny.

4.5 Aplikační rozhraní CAN

Aplikační rozhraní CAN poskytuje podobně jako aplikační rozhraní USB služby vyšší aplikační vrstvě. Reprezentuje sběrnici CAN z obecného pohledu - využití nezávisí na znalosti implementace nižších vrstev nebo znalosti práce se samotnou hardwarovou periferií.

Na rozdíl od aplikačního rozhraní USB definuje aplikační rozhraní CAN dvě funkce, které nejsou součástí standardu CAN. Jedná se o simulaci více zařízení sběrnice a podporu časování rámců.

Implementace jednotlivých funkcí aplikačního rozhraní CAN probíhala především s cílem splnit požadavek zadání diplomové práce respektovat protokol existujícího ovladače pro operační systém Microsoft Windows XP.

4.5.1 Inicializace rozhraní

Inicializace aplikačního rozhraní se provádí voláním funkce *CAN_iface_init*. Funkce povoluje potřebné periferie, nastavuje vstupně-výstupní piny potřebné pro provoz periferie CAN, konfiguruje přerušení periferie CAN, konfiguruje objekty zpráv pro příjem a odesílání rámců a inicializuje fronty rámců.

4.5.2 Nastavení rozhraní

Z důvodu změn nastavení sběrnice CAN za běhu systému se aplikační rozhraní CAN na rozdíl od rozhraní USB nenastavuje prostřednictvím konfiguračních souborů, ale prostřednictvím sady funkcí. Výjimku tvoří nastavení velikostí front rámců a stavů, které lze nastavit v souboru *can_iface_defines.h*. Význam front rámců a stavů bude objasněn v následujících kapitolách.

Funkce *CAN_iface_set_bitrate* konfiguruje rychlost sběrnice CAN a nastavuje, zda se bude periferie snažit znovu automaticky posílat rámce, které se nepodařilo odeslat. Po dohodě s vedoucím práce byly implementovány pouze rychlosti sběrnice CAN definované knihovnou *STR71x firmware library*: 100 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s a 1000 kbit/s. Tyto rychlosti reprezentují množinu nejčastěji používaných rychlostí sběrnice CAN.

Pro nastavení testovacích módů periferie CAN, které jsou popsány v kapitole 3.4.5.1 „Testovací módy periferie CAN“, slouží funkce *CAN_iface_set_test_mode*. Pomocí této funkce lze povolit nebo zakázat testovací mód periferie a případně nastavit loopback mód, silent mód nebo jejich kombinaci.

4.5.3 Příjem rámců

Pokud je aplikační rozhraní CAN správně konfigurováno, umožňuje aplikační vrstvě číst rámce sběrnice CAN, které poslaly ostatní uzly nebo zařízení samotné. Pokud dojde na sběrnici CAN k přenosu rámce, informuje o tomto přenosu aplikační rozhraní knihovna CAN pomocí funkce zpětného volání aplikačního rozhraní. Aplikační rozhraní získá prostřednictvím parametrů funkce zpětného volání čas, ve kterém došlo k přijetí rámce, typ rámce, identifikátor rámce a data přenášeného rámce. Vše pak uloží v podobě speciální struktury do fronty příchozích rámců.

Fronta příchozích rámců je realizována prostřednictvím kruhového bufferu, jehož velikost lze změnit v souboru *can_iface_defines.h*.

Funkce *CAN_iface_read_recv_msg* aplikačního rozhraní je určena pro čtení rámců uložených ve frontě příchozí rámců. Její návratová hodnota je výčtový typ, který informuje, zda je, či není ve frontě příchozích rámců nějaký přijatý rámec. Případný přijatý rámec je předán aplikační vrstvě v podobě parametrů zmíněné funkce a jeho místo ve frontě je uvolněno.

Stejně jako aplikační rozhraní USB i aplikační rozhraní periferie CAN disponuje funkcí *CAN_iface_register_callbacks*. Pomocí této funkce lze registrovat funkci zpětného volání, která je použita při přijetí rámce periferií CAN a zajišťuje tak okamžitou odezvu pro aplikační vrstvu při příjmu rámce.

4.5.4 Odesílání rámců

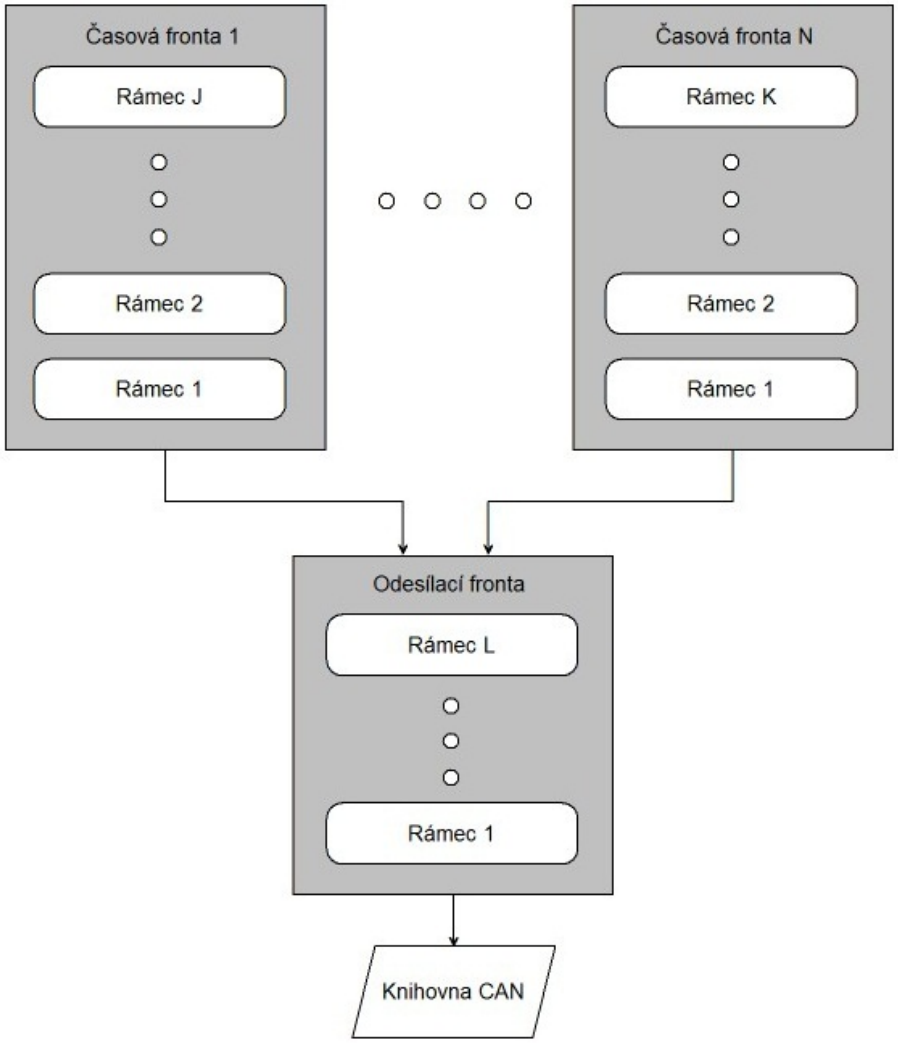
Pro odesílání rámců na sběrnici CAN je využíván systém front, jehož struktura je znázorněna na obrázku 4.5. Funkcí *CAN_iface_send_can* aplikační vrstva požádá aplikační rozhraní CAN o odeslání rámce. Parametry této funkce udávají číslo fronty pro odeslání, typ rámce, identifikátor rámce, data rámce a čas, kdy má dojít k jeho odeslání. Předpokládá se, že rámce jsou do jednotlivých časových front vkládány již seřazeny podle času jejich odeslání, počínaje rámcem, který má být odeslán jako první.

Po vložení nového rámce do některé z časových front je aktualizován časový alarm, který vyvolá přerušení v čase, odpovídajícímu nejnižšímu z časů všech rámců ve všech časových frontách. Při přerušení alarmu dochází k přesunu všech rámců, které mají být odeslány v aktuální čas, do odesílací fronty. Rámce v odesílací frontě jsou řazeny podle hodnoty svých identifikátorů od nejnižšího po nejvyšší, což zaručuje přednostní odeslání rámců s vyšší prioritou. Po přesunu rámců do odesílací fronty je opět aktualizována hodnota alarmu. Pokud jsou časové fronty prázdné, je alarm deaktivován.

Rámce v odesílací frontě jsou postupně odesílány na sběrnici CAN prostřednictvím funkce knihovny CAN. Pokud je aplikační rozhraní požádáno o odeslání rámce s časem, který je nižší než aktuální čas, je tento rámec zařazen přímo do odesílací fronty.

Jednotlivé časové fronty mají za úkol simulovat samostatné uzly na sběrnici CAN.

Rámec je po odeslání na sběrnici přesunut z odesílací fronty do fronty odeslaných rámců společně s časem, kdy došlo skutečně k přenosu tohoto rámce na sběrnici. Pokud dojde k chybě při odesílání rámce, je takový rámec opět přesunut do fronty odeslaných rámců s příznakem chyby při přenosu.



Obrázek 4.5: Struktura front pro odesílání rámců CAN

Funkce *CAN_iface_read_sent_msg* umožňuje vyčítat rámce z fronty odeslaných rámců. Jako v předchozích případech informuje návratová hodnota funkce, zda fronta obsahuje nějaký rámec k přečtení. Údaje případného rámce jsou poté předány aplikační vrstvě prostřednictvím parametrů zmíněné funkce.

Stejně jako u ostatních událostí aplikačních rozhraní, je možné pro událost úspěšného odeslání rámce registrovat funkci zpětného volání. Toho lze opět docílit pomocí funkce *CAN_iface_register_callbacks*.

Všechny zmíněné fronty jsou implementovány prostřednictvím kruhového bufferu. Parametry front, týkající se velikostí front a počtu časových front, lze měnit v souboru *can_iface_defines.h*.

4.5.5 Stavy uzlu CAN

Periferie CAN může přecházet mezi různými stavy, jak je popsáno v kapitole 3.3.3.3 „Chybové stavy uzlů“. Při přechodu do nového stavu je informace uložena do fronty stavů sběrnice, kterou aplikační vrstva vyčítá pomocí funkce *CAN_iface_read_state*. Aplikační vrstva získává touto funkcí informaci o novém stavu uzlu, o čase, kdy došlo k přechodu do tohoto stavu, a kód chyby uzlu[1].

Pro okamžitou odezvu na přechod uzlu do nového stavu je možné využít funkci *CAN_iface_register_callbacks* pro registraci funkce zpětného volání při výskytu této události.

Pokud aplikační vrstva vyžaduje informaci o aktuálním stavu uzlu (ne pouze o přechodu mezi stavy), může využít funkci *CAN_iface_get_state*, jejíž návratová hodnota obsahuje požadovanou informaci.

4.6 Knihovna CAN

Knihovna CAN poskytuje aplikačnímu rozhraní CAN prostředky pro práci s konkrétní periferií CAN, kterou obsahuje hardwarový přípravek. Knihovna CAN odděluje logiku aplikačního rozhraní od hardwarové implementace řadiče CAN.

Vývojář aplikační vrstvy volá pouze funkce aplikačního rozhraní CAN. Funkce knihovny CAN náleží nižším vrstvám.

4.6.1 Konfigurace periferie CAN

Funkce *CAN_enable_periphetal* knihovny CAN povoluje periferie CAN a vstupně-výstupní brány potřebné pro komunikaci periferie. *CAN_setup_gpios* konfiguruje zmíněné vstupně-výstupní brány.

Funkce *CAN_set_bitrate* konfiguruje rychlost sběrnice CAN a nastavuje, zda se bude periferie snažit znovu automaticky posílat rámce, které se nepodařilo odeslat. Tato funkce je volána funkcí *CAN_iface_set_bitrate*, která tím funkci *CAN_set_bitrate* poskytuje aplikační vrstvě. Funkce zároveň restartuje periferii a uvádí ji do aktivního stavu. Lze ji tedy použít pro přechod ze stavu „odpojené zařízení“.

Funkce *CAN_init_message_objects* inicializuje objekty zpráv periferie CAN, které slouží pro příjímání a odesílání rámců sběrnice CAN. Mechanismus funkce objektů zpráv shrnuje kapitola 3.4.5 „Periferie CAN“.

4.6.2 Odesílání rámců na sběrnici CAN

Pro odesílání rámců na sběrnici CAN poskytuje knihovna CAN funkci *CAN_send_message*. Funkci je předán identifikátor rámce, DLC rámce a data rámce. Funkce poté na základě typu rámce vyhodnotí, který objekt zpráv využije pro odeslání rámce, zkopíruje příslušná data do vybraného objektu zpráv a zajistí odeslání rámce na sběrnici.

[1] Význam jednotlivých bitů kódu chyby uzlu definuje tabulka 3.20.

Informace o úspěchu nebo neúspěchu a skutečný čas odeslání rámce lze získat pomocí obsluhy přerušení periferie CAN popsané v následující kapitole.

4.6.3 Obsluha přerušení periferie CAN

Pro konfiguraci přerušení periferie CAN slouží funkce *CAN_setup_interrupt*, jejímž jediným parametrem je priorita přerušení, která má být tomuto přerušení nastavena.

Pokud je detekováno přerušení od periferie CAN, je volána funkce *CAN_interrupt_handler*. Funkce rozpozná typ přerušení a adekvátně na něj zareaguje. Přerušení může nastat při přijetí rámce, odeslání rámce nebo přechodu periferie do nového stavu.

Při přijetí rámce kopíruje funkce data rámce z periferie a společně s časem přijetí rámce předává tato data aplikačnímu rozhraní CAN. Při úspěšném odeslání rámce informuje funkce o této události aplikační rozhraní CAN, které může odeslaný rámec přesunout z odesílací fronty do fronty odeslaných rámců. Stejně tak při přechodu periferie do nového stavu informuje funkce aplikační rozhraní o tomto novém stavu.

Při implementaci obsluhy přijatých rámců se nedařilo správně přijímat rámce typu žádost o data. Periferie CAN tyto rámce vyhodnocovala jako datové rámce s nulovou délkou datové oblasti. Po konzultaci s vedoucím práce byla tato dysfunkce ignorována. Vyšší vrstvy modelu jsou přesto připraveny na možnost, že se v budoucnu (např. na jiném hardwaru) podaří rozpoznávat přijaté rámce typu žádost o data.

4.7 Aplikační vrstva

Tato kapitola má za úkol vysvětlit práci s aplikační vrstvou pomocí jednoduchých příkladů různých aplikačních vrstev. Poslední podkapitola popisuje implementaci protokolu existujícího ovladače, který je popsán v kapitole 3.5 „Protokol USB existujícího ovladače“.

Aplikační vrstva implementuje chování hardwarového přípravku. Využívá služeb poskytovaných aplikačními rozhraními periferií USB a CAN. Díky relativně jednoduchému použití aplikačních rozhraní je i pro nezkušeného vývojáře firmwaru snadné vytvořit vlastní aplikační vrstvu.

Aplikační vrstva je tvořena zdrojovým souborem a k němu příslušným hlavičkovým souborem, který je vložen do standardních zdrojových souborů[2] *main.c* a *71x_it.c* pomocí direktivy jazyka C (kód 4.1).

Kód 4.1: Vložení aplikační vrstvy

```
1 #include "jmeno_hlavickoveho_souboru.h"
```

Jediným nutným požadavkem na aplikační rozhraní je implementace dvou funkcí: *app_init* a *main_loop*.

[2]Standardními zdrojovými soubory jsou myšleny zdrojové soubory, které se při implementaci aplikační vrstvy nemění a jsou pevnou součástí systému.

Funkce *app_init* je volána vždy jednou při spuštění firmwaru přípravku. Při volání této funkce jsou již inicializována aplikační rozhraní USB a CAN, ale ještě nejsou povolena přerušení příslušných periferií. V této funkci tedy lze provádět různé inicializace, které souvisejí se samotnou funkcí aplikační vrstvy.

Funkce *main_loop* je cyklicky volána při samotném běhu aplikace. Představuje tedy hlavní smyčku firmwaru přípravku. Její běh je přerušován pouze přerušeními jednotlivých periferií. V této funkci lze vykonávat základní funkcionalitu aplikační vrstvy.

Minimální aplikační vrstvu tvoří šablona složená ze dvou souborů, které nemají žádnou funkcionalitu. Jejich obsah je zachycen kódy 4.2 a 4.3.

Kód 4.2: Hlavičkový soubor minimální aplikační vrstvy

```
1 /* app_layer.h */
2 #ifndef __APP_LAYER_H
3 #define __APP_LAYER_H
4
5 void app_init(void);
6 void main_loop(void);
7
8 #endif
```

Kód 4.3: Zdrojový soubor minimální aplikační vrstvy

```
1 /* app_layer.c */
2 #include "app_layer.h"
3
4 void app_init()
5 {}
6
7 void main_loop()
8 {}
```

4.7.1 Příklad implementace aplikační vrstvy

Jednoduchou aplikací může být například posílání dat přijatých periferií USB na sběrnici CAN s předem definovaným identifikátorem. Tato aplikace tak může například simulovat senzor automobilu, v němž simulovaná data senzoru poskytuje aplikace PC prostřednictvím sběrnice USB.

Výše popsána funkcionalita je implementována aplikační vrstvou, která je realizována kódem 4.4.

Řádky 4 až 8 definují postupně rychlost sběrnice CAN, číslo časové fronty pro odesílaný rámec, typ rámce (nula pro standardní rámec, jedna pro rozšířený rámec), hodnotu identifikátoru rámce a maximální velikost dat rámce CAN.

V inicializační funkci proběhne po spuštění přípravku konfigurace rychlosti sběrnice CAN, povolení automatického opakování posílání rámců, které se nepodařilo odeslat, a zakázání zpětného příjmu odeslaných rámců (funkce *CAN_iface_set_bitrate* na řádku č. 15).

V hlavní smyčce je poté testována návratová hodnota funkce pro čtení dat přijatých na USB, tedy příjem dat na některém z koncových bodů (řádek č. 20). Pokud byla data na některém z koncových bodů přijata, je testováno, zda jejich velikost nepřekračuje maximální možnou velikost datové části rámce CAN (řádek č. 22), a pokud ne, jsou odeslána na sběrnici CAN (řádek č. 23). Nastavením času odeslání rámce na nulovou hodnotu zajistíme okamžité odeslání rámce a poté uvolníme příchozí buffer aplikačního rozhraní USB (řádek č. 25).

Kód 4.4: Příklad simulace senzoru

```
1  /* app_layer.c */
2  #include "app_layer.h"
3
4  #define CAN_BITRATE      500
5  #define FRM_QUEUE        0
6  #define FRM_TYPE         0
7  #define FRM_IDENT        0x12
8  #define CAN_MAX_SIZE 8
9
10 u32 size;
11 u8 * data;
12
13 void app_init()
14 {
15   CAN_iface_set_bitrate(CAN_BITRATE, 1, 0);
16 }
17
18 void main_loop()
19 {
20   if (USB_iface_read_recv_msg(0, &size, &data))
21   {
22     if (size<=CAN_MAX_SIZE)
23       CAN_iface_send_can(FRM_QUEUE, FRM_TYPE, FRM_IDENT, 0, size,
           data);
24
25     USB_iface_release_recv_msg();
26   }
27 }
```

Kód 4.5 demonstruje použití funkce zpětného volání při příjmu dat na USB. V inicializační funkci přibyla registrace funkce pro zpětné volání při příjmu dat na USB (řádek č. 24). Hlavní smyčka zůstává prázdná a tím se nezatěžuje procesor opakovaným dotazováním, jestli byla

data na USB přijata. Při příjmu dat na USB je automaticky volána funkce *recv_usb*, jejíž význam je zřejmý z předchozího příkladu.

Kód 4.5: Příklad simulace senzoru s využitím zpětného volání

```
1  /* app_layer.c */
2  #include "app_layer.h"
3
4  #define CAN_BITRATE      500
5  #define FRM_QUEUE        0
6  #define FRM_TYPE         0
7  #define FRM_IDENT        0x12
8  #define CAN_MAX_SIZE 8
9
10 void recv_usb()
11 {
12   u32 size;
13   u8 * data;
14
15   USB_iface_read_recv_msg(0, &size, &data))
16
17   if (size<=CAN_MAX_SIZE)
18       CAN_iface_send_can(FRM_QUEUE, FRM_TYPE, FRM_IDENT, 0, size,
           data);
19
20   USB_iface_release_recv_msg();
21 }
22
23 void app_init()
24 {
25   USB_iface_register_callbacks(recv_usb, 0);
26   CAN_iface_set_bitrate(CAN_BITRATE, 1, 0);
27 }
28
29 void main_loop()
30 {
31 }
```

4.7.2 Implementace existujícího protokolu

Jedním z požadavků zadání diplomové práce bylo implementovat protokol převodníku USB/-CAN existujícího ovladače operačního systému Microsoft Windows XP. Protokol je podrobně popsán v kapitole 3.5 „Protokol USB existujícího ovladače“.

Existující protokol je implementován aplikační vrstvou tvořenou soubory *app_layer_CANexpl.h* a *app_layer_CANexpl.c*. Hlavičkový soubor *app_layer_CANexpl.h* zbavený komentářů je znázorněn kódem 4.6. Soubor *app_layer_CANexpl.c*, který implementuje jednotlivé funkce hlavičkového souboru, zde není pro svůj velký rozsah uveden.

Pro demonstraci obou způsobů, jakými lze reagovat na události jednotlivých periferií (opakované dotazování, zda nedošlo k události, nebo mechanismus zpětného volání), obsahuje aplikační vrstva speciální definici (kód 4.6, řádek č. 10). Pokud je tato definice v hlavičkovém souboru aplikační vrstvy ponechána, bude aplikační vrstva reagovat na události periferií pomocí funkcí zpětného volání. Pokud dojde k odstranění této definice, bude ve funkci *main_loop* docházet k opakovanému dotazování na výskyt událostí aplikačních rozhraní. Oba způsoby implementace jsou z hlediska správného fungování aplikační vrstvy totožné.

Kód 4.6: Soubor app_layer_CANexpl.h zbavený komentářů

```
1  #ifndef __APP_LAYER_CANEXPL_H
2  #define __APP_LAYER_CANEXPL_H
3
4  #include "debug.h"
5  #include "71x_type.h"
6  #include "usb_iface_config.h"
7  #include "can_iface.h"
8  #include "usb_iface.h"
9
10 #define APP_USE_CALLBACKS
11
12 extern USB_IFACE_EP_IN USB_iface_ep_ins [USB_IFACE_EP_IN_NUM];
13 extern USB_IFACE_EP_OUT USB_iface_ep_outs [USB_IFACE_EP_OUT_NUM];
14
15 void app_init();
16 void main_loop();
17
18 void timer_overflow();
19 void recieve_can(void);
20 void sent_can(void);
21 void can_state(void);
22 void recieve_usb(void);
23 void sent_usb(void);
24
25 #endif
```

Pokud je povolena definice pro použití funkcí zpětného volání, dochází ve funkci *app_init* k registraci funkcí zpětného volání periferie CAN a USB. Dále je v této funkci registrováno zpětné volání pro přetečení časovače rámců sběrnice CAN.

Jak bylo popsáno v kapitole 3.5 „Protokol USB existujícího ovladače“, komunikace zařízení s PC probíhá po jednom koncovém bodě typu OUT (směr z PC do zařízení) a po dvou

koncových bodech typu IN (směr ze zařízení do PC).

Koncové body IN slouží zařízení pro předání informace do PC. Jeden koncový bod typu IN je určen pro blokový přenos a druhý pro přerušení. Data pro koncový bod přerušení jsou udržována ve struktuře jazyka C, čímž lze snadno a přehledně měnit hodnoty jednotlivých položek přenášených dat. Zprávy protokolu pro koncový bod blokového přenosu jsou ukládány do interního bufferu aplikační vrstvy. Aplikační vrstva zároveň udržuje informaci o počtu zpráv, které zmíněný buffer obsahuje.

V hlavní smyčce aplikace *main_ lopp* je testováno, zda jsou v interním bufferu pro blokový přenos data k odeslání. Pokud ano, je informace o velikosti dat v bufferu a počtu zpráv aktualizována ve struktuře pro koncový bod přerušení. Následuje odeslání dat přerušení a dat blokového přenosu.

Při úspěšném odeslání dat je volána funkce *sent_ usb*. Funkce rozpozná, zda došlo k odeslání dat z koncového bodu pro blokový přenos, a pokud ano, je vyprázdněn interní buffer pro tento koncový bod a aktualizovány hodnoty struktury přerušení.

Koncový bod typu OUT je určen pro zasílání příkazů do zařízení. Pro příjem a obsluhu těchto příkazů slouží funkce *recieve_ usb* definovaná na řádku 22 kódu 4.6.

Funkce v prvním kroku zkontroluje, zda jsou na USB nějaká příchozí data, a pokud ne, funkce končí. Tato podmínka by nebyla nutná, pokud by se využívalo pouze mechanismu zpětného volání, který funkci vykoná vždy při příjmu dat na USB.

Pokud jsou periferií USB přijata data, funkce *recieve_ usb* kontroluje, zda není přeplněn interní buffer určený pro data odpovědí na příchozí příkazy, a pokud ano, je volána funkce, která prostřednictvím koncového bodu přerušení informuje PC o plném USB bufferu zařízení.

Následuje analýza přijatých dat. Funkce rozpozná typ příkazu a adekvátně na něj reaguje. Pokud je z definice existujícího protokolu nutné odpovídat na příchozí příkaz, je příslušná odpověď zapsána do interního bufferu a je aktualizována hodnota počtu zpráv v tomto bufferu. Při chybě v průběhu analýzy přijatých dat je volána lokální funkce pro odeslání chyby protokolu USB prostřednictvím koncového bodu přerušení.

Funkce *recieve_ usb* končí uvolněním bufferu přijatých dat aplikačního rozhraní USB.

Příjem dat na sběrnici CAN realizuje funkce *recieve_ can*. Funkce postupně vyčítá rámce přijaté na sběrnici CAN pomocí funkce aplikačního rozhraní CAN *CAN_ iface_ read_ recv_ msg*. Po načtení rámce do lokálních proměnných dojde k rozpoznání typu příchozího rámce a následnému uložení zprávy protokolu USB obsahující příchozí rámec CAN do interního bufferu pro blokový přenos. Jak již bylo řečeno, odesílání zpráv interního bufferu je realizováno ve funkci *main_ loop*.

Rámce se na sběrnici CAN zasílají prostřednictvím příkazů USB, které obstarává výše zmíněná funkce *recieve_ usb*. Při úspěšném odeslání rámce na sběrnici je volána funkce aplikační vrstvy *sent_ usb*. V této funkci dochází k aktualizaci položek struktury pro koncový bod přerušení, které informují o počtu rámců v odesílacích frontách.

Funkce *timer_ overflow* je volána při přetečení časovače, který udržuje hodnotu aktuálního času pro odesílání rámců CAN. Funkce připojí do interního bufferu pro blokový přenos zprávu protokolu USB o přetečení časovače.

Dojde-li ke změně stavu periferie CAN, je volána funkce *can_ state*, která do interního bufferu pro blokový přenos připojí zprávu o novém stavu periferie CAN. Pokud je novým

stavem periferie CAN odpojený stav a je povolena automatická obnova z odpojeného stavu, je periferie CAN uvedena do aktivního stavu.

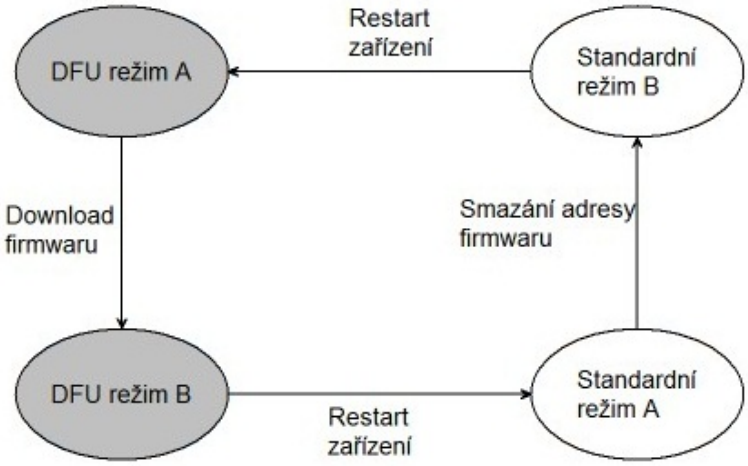
Konzistence dat je v aplikační vrstvě řešena systematickým vypínáním a zapínáním přerušení, jejichž obsluha může tuto konzistenci narušit.

4.8 Implementace DFU

Hardwarový přípravek může pracovat ve dvou režimech: standardní režim a DFU režim. Oba režimy jsou implementovány nezávislými firmware aplikacemi.

Režim DFU je určen pro aktualizaci firmwaru standardního režimu. Mechanismus, jakým dochází k aktualizaci firmwaru pomocí DFU, popisuje kapitola 3.6 „Direct Firmware Update (DFU)“.

Implementace firmwaru režimu DFU vychází z příkladu, který obsahuje knihovna „STR7-STR9 USB developer kit“. Pro správnou funkci firmwaru bylo nutné pochopit fungování existujícího příkladu a provést v jeho kódu několik změn.



Obrázek 4.6: Přechodový diagram režimů zařízení

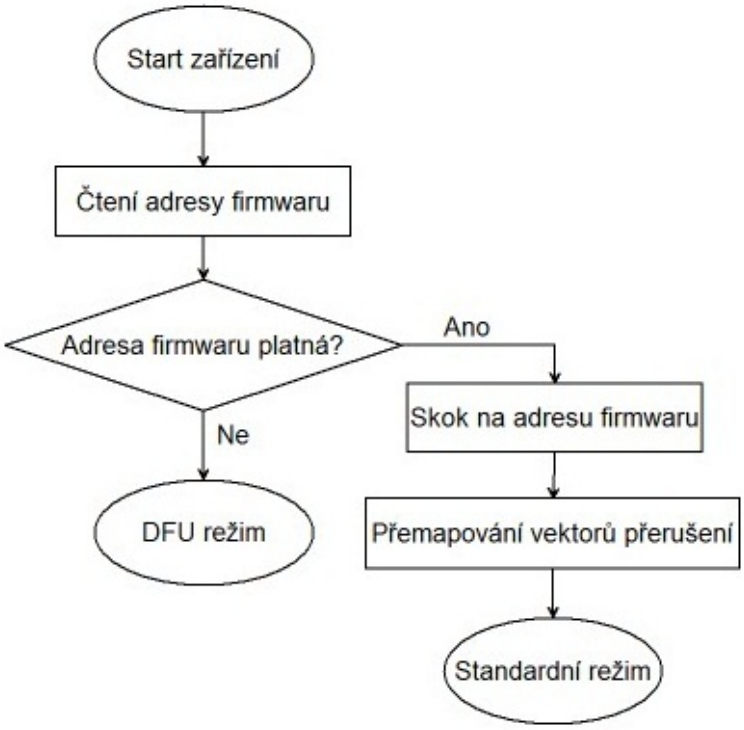
Obrázek 4.6 demonstruje podmínky, za kterých dochází k přechodu mezi režimem DFU a standardním režimem.

Vycházejme ze stavu *DFU režim A*. Tento stav je normálním režimem DFU, ve kterém se zařízení chová tak, jak je popsáno v kapitole 3.6. Při přenosu nového firmwaru pro standardní režim dochází k přechodu zařízení do stavu *DFU režim B*. Stav *DFU režim B* je totožný se stavem *DFU režim A* s jediným rozdílem: po restartu zařízení, které se nachází ve stavu *DFU režim B*, se bude zařízení nacházet ve stavu *Standardní režim A*, který je realizován novým firmwarem pro standardní režim.

Standardní režim A je implementován aplikační vrstvou popsanou v předešlých kapitolách. Pokud dojde k volání funkce *USB_iface_enter_DFU* aplikačního rozhraní USB, dojde ke smazání adresy firmwaru v paměti flash a přechodu zařízení do stavu *Standardní režim B*. Podobně jako DFU režimy A a B se standardní režimy A a B liší pouze skutečností, že

při restartu zařízení, které se nachází ve stavu *Standardní režim B*, přejde zařízení do stavu *DFU režim A*.

Restartu zařízení lze dosáhnout pomocí tlačítka umístěného na plošném spoji hardwarového přípravku nebo odpojením a opětovným připojením napájení hardwarového přípravku (přípravek je napájen prostřednictvím kabelu USB). Výhoda druhého způsobu je, že pro restartování, které je potřebné pro přechod mezi režimy, není nutné demontovat ochranný kryt přípravku.



Obrázek 4.7: Zavedení režimu po startu zařízení

Zavedení správného režimu při startu zařízení obstarává logika znázorněná na obrázku 4.7. Po startu zařízení je vždy spuštěn firmware DFU režimu. Ještě před inicializací a spuštěním samotné funkce DFU je přečteno slovo na adrese 0x40008000 paměti flash, které reprezentuje adresu v paměti přípravku, na které leží počátek strojového kódu firmwaru standardního režimu. Pokud jsou všechny bity přečtené adresy stejné (hodnota slova je 0x0000 nebo 0xFFFF), je adresa prohlášena za neplatnou a přípravek pokračuje ve vykonávání firmwaru režimu DFU. V opačném případě je vykonávání firmwaru režimu DFU přerušeno a je proveden skok na počátek strojového kódu standardního režimu. Před samotným spuštěním standardního režimu musí dojít v inicializaci režimu k přemapování vektorů přerušení na počátek paměti RAM.

4.8.1 Generování obrazu DFU

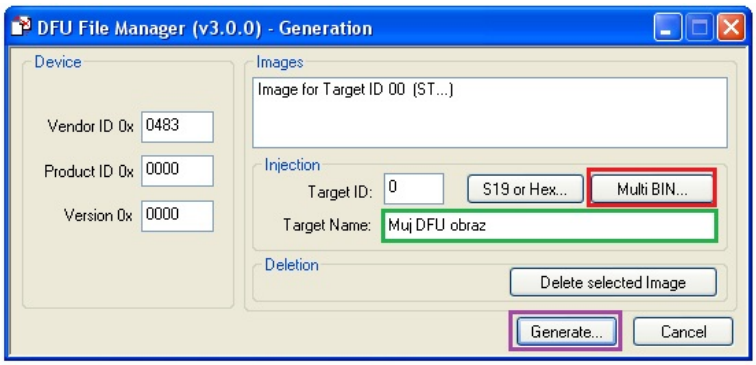
Tato kapitola popisuje krok po kroku postup, jakým lze generovat obrazy DFU z binárních souborů firmwaru.

V prvním kroku je nutné instalovat sadu nástrojů pro DFU, které poskytuje samotný výrobce procesoru STMicroelectronics. Instalační soubor této sady nástrojů naleznete na CD diplomové práce v souboru „DfuSe.zip“.

Dalším krokem je vygenerování binárního souboru BIN firmwaru, který chceme nahrát pomocí DFU do přípravku. Před kompilací a generováním souboru BIN je nutné správně nastavit adresu paměti, na kterou bude firmware nahrán. V projektu přiloženém na CD je tato adresa nastavena na 0x40010000. Doporučuji toto nastavení neměnit. Na této adrese se nachází sektor velikosti 64Kb, který je svým rozsahem dostatečně velký pro potřeby převodníku. Při změně adresy by bylo zároveň nutné přemapovat adresy vektorů přerušení.

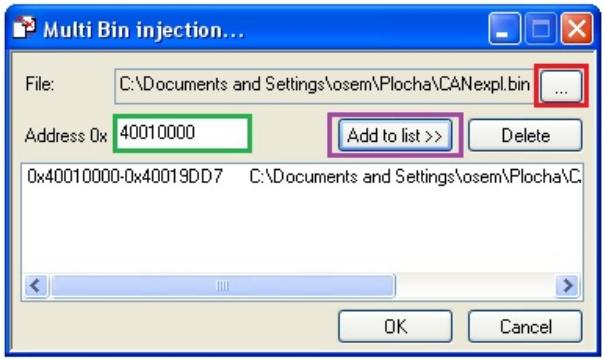
Soubor BIN je nutné převést na obraz DFU, což lze provést pomocí nástroje „DFU File Manager“, který je součástí zmíněné sady nástrojů.

Po spuštění nástroje je uživatel dotázán na akci, kterou chce vykonat. Vybereme první možnost „I want to GENERATE a DFU file from S19, HEX or BIN files“ a pokračujeme stiskem tlačítka „OK“.



Obrázek 4.8: Dialog generování DFU obrazu

Zobrazí se dialog zachycený na obrázku 4.8. Klikneme na tlačítko „Multi BIN...“ označené na obrázku červeným rámečkem.



Obrázek 4.9: Dialog výběru BIN souboru

V následném dialogu „Multi Bin injection...“ (obrázek 4.9) klikneme na tlačítko označené červeným rámečkem a vybereme námi vygenerovaný BIN soubor. Do pole označeného zele-

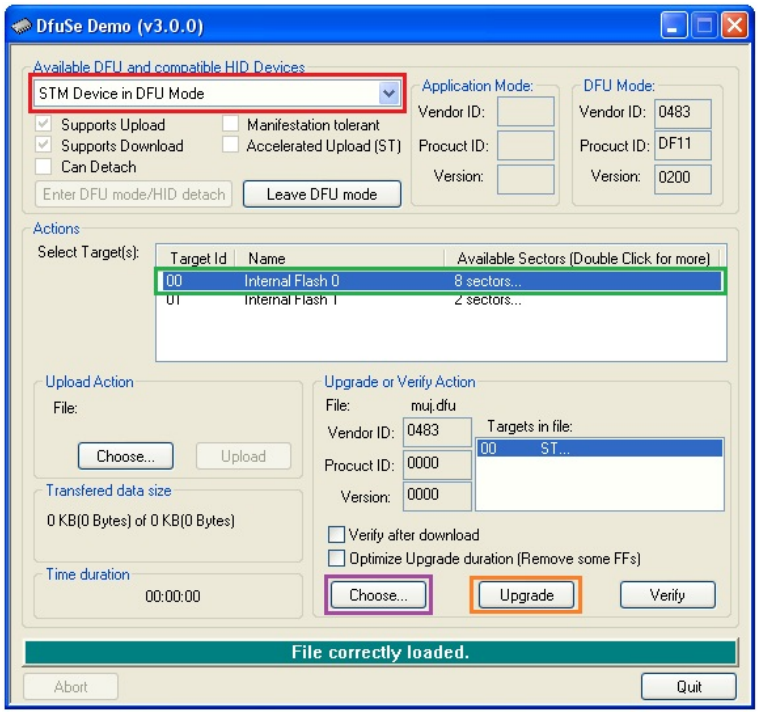
ným rámečkem napíšeme adresu paměti, do které chceme požadovaný firmware nahrát (pro náš příklad 40010000). Stisknutím tlačítka „Add to list »“ přidáme BIN soubor do obrazu DFU a vše potvrdíme tlačítkem „OK“.

Zpět v dialogu 4.8 můžeme zvolit název obrazu a zapsat ho do pole označeného na obrázku zeleným rámečkem.

Generování obrazu spustíme tlačítkem „Generate...“, které po vyzvání k zadání názvu a umístění DFU souboru obsahujícího DFU obraz vygeneruje příslušný soubor obrazu.

4.8.2 Aktualizace firmwaru

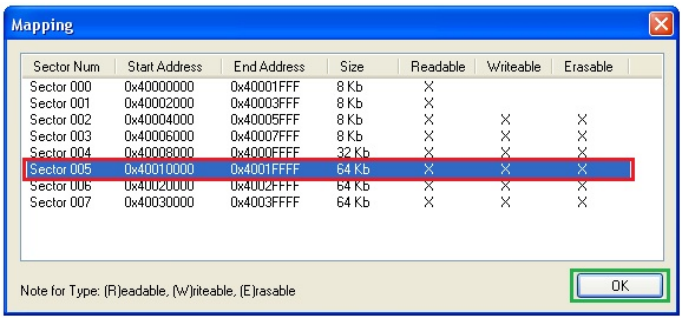
Přenos obrazu DFU, který jsme vygenerovali v předchozí kapitole, provedeme pomocí nástroje „DfuSe Demo“. Nástroj „DfuSe Demo“ je součástí sady nástrojů zmíněné v předchozí kapitole.



Obrázek 4.10: Dialog nástroje DfuSe Demo

Připojíme zařízení v režimu DFU. Pokud připojujeme přípravek do operačního systému poprvé, je možné, že bude operační systém vyžadovat ovladače zařízení DFU. Tyto ovladače nalezneme ve složce, do které jsme nainstalovali sadu nástrojů pro DFU (zpravidla „C:\ProgramFiles\STMicroelectronics\Software\DfuSe\Driver“).

Po spuštění nástroje „DfuSe Demo“ se zobrazí dialog na obrázku 4.10. Z nabídky v červeném rámečku vybereme připojené DFU zařízení, do kterého chceme nahrát DFU obraz. Dvojitým poklepáním na řádek označený rámečkem zelené barvy otevřeme dialog 4.11 sloužící pro výběr sektoru, do kterého bude nahrán obraz DFU. V tomto dialogu vybereme



Obrázek 4.11: Dialog mapování obrazu nástroje DfuSe Demo

sektor 005, který odpovídá adrese 0x40010000. Potvrzení výběru sektoru a návrat do původního dialogu provedeme stiskem tlačítka „OK“. V původním dialogu vybereme tlačítkem „Choose...“ soubor s obrazem DFU. Download obrazu do zařízení zahájíme stiskem tlačítka „Upgrade“. Mechanismus DFU provede postupně smazání námi vybraného sektoru a stažení nového firmwaru do zařízení.

Při dalším restartu zařízení bude aktivován nový firmware, který jsme do zařízení nahráli pomocí obrazu DFU.

4.9 Úprava existujícího ovladače a knihovny

V kapitole 3.6 „Direct Firmware Update (DFU)“ je uvedeno, že změna módu zařízení musí být vyvolána vnějším podnětem. Pro plnou podporu mechanismu DFU tedy bylo nutné přidat do existujícího protokolu USB příkaz, v rámci kterého bude aplikační vrstva volat funkci *USB_iface_enter_DFU* aplikačního rozhraní USB. Funkce *USB_iface_enter_DFU* slouží pro přechod ze standardního režimu do režimu DFU. Zároveň bylo nutné doplnit knihovnu využívající tento ovladač o funkci, která bude zprostředkovávat zaslání tohoto příkazu běžným uživatelským aplikacím.

Podrobný popis implementace původní verze ovladače je obsažen v kapitole „Programování ovladačů“ diplomové práce [12]. Po nastudování struktury zdrojových kódů jsem do ovladače přidal definici *IOCTL_DFU_MODE*, která reprezentuje ioctl požadavek na zaslání příkazu pro přechod do DFU módu. Následně jsem do kódu ovladače doplnil potřebnou obsluhu při volání funkce ovladače, která reaguje na různé kódy ioctl.

Do knihovny pro upravený ovladač jsem doplnil funkci *Enter_Dfu_Mode*, která vyvolá žádost ioctl ovladače na přechod do režimu DFU.

Struktura příkazu doplněného do existujícího protokolu USB je patrná z tabulky 4.3.

	B0
OUT	'X'

Tabulka 4.3: Příkaz pro přechod do režimu DFU

4.10 Ladění firmwaru

Použitý mikrokonrolér podporuje ladění firmwaru pomocí rozhraní JTAG (Joint Test Action Group). Rozhraní JTAG je definováno normou IEEE 1149.1 [8]. Primárním účelem tohoto rozhraní je programování flash pamětí a podpora ladění firmwaru za běhu. Při použití rozhraní JTAG lze ve vývojovém prostředí nastavit body přerušení běhu programu přímo ve zdrojovém textu, krokovat program po jednotlivých příkazech jazyka C nebo strojového kódu a prohlížet aktuální obsah proměnných a registrů v přípravku.

Běh některých částí firmwaru nelze přerušit bodem přerušení, aniž by došlo k nějaké chybě. Příkladem může být enumerace zařízení USB, kdy je nutné odpovídat na dotazy a příkazy USB hostitele do předem definované doby. Zastavením běhu programu v průběhu enumerace dojde k časové prodlevě odpovědi a enumerace zařízení se nezdaří. Z tohoto důvodu implementuje práce množinu funkcí, které jsou schopny zasílat data na sériové rozhraní UART. Tyto funkce jsou definovány v souborech *debug.h* a *debug.c* a podporují zasílání jednotlivých znaků, řetězců jazyka C a hexadecimální hodnoty různých číselných proměnných.

Kapitola 5

Testování

K testování funkcí jednotlivých modulů docházelo jak v průběhu vývoje, tak po dokončení kompletního firmwaru. V průběhu vývoje bylo k testování využito sériového výstupu UART a rozhraní JTAG.

Testování se zaměřilo na test systému s aplikační vrstvou, která implementuje existující protokol USB.

5.1 Testování s aplikací CAN Explorer

K hlavní části testování byl použit program „CAN Explorer“, který využívá knihovnu ovladače pro operační systém Microsoft Windows XP, a tudíž komunikuje s přípravkem pomocí existujícího protokolu USB. Docházelo tak jak k testování implementace existujícího protokolu v aplikační vrstvě přípravku, tak zároveň k testování správného fungování převodníku.

Program „CAN Explorer“ je GUI aplikace pro operační systém Microsoft Windows XP, která umožňuje konfigurovat sběrnici CAN a vysílat a přijímat rámce sběrnice CAN. V rámci konfigurace sběrnice CAN prostřednictvím zmíněného programu lze nastavit i loopback mód periferie CAN hardwarového přípravku.

V první fázi probíhalo testování v **loopback módu** periferie CAN. V tomto módu byly rámce vysílané na sběrnici CAN zpětně přijímány přípravkem, což potvrdilo správnou funkci přípravku nastaveného do loopback módu.

Další fáze testování probíhala na sběrnici CAN s **řídicí jednotkou airbagu automobilu** „Airbag VW51“. Přípravek bez problémů přijímal jím zasílaná data. Při odeslání dat na sběrnici mu bylo odeslání dat řadičem CAN airbagu správně potvrzeno. Tato fáze testování tedy proběhla také bez potíží.

Poslední fáze testování pomocí aplikace „CAN Explorer“ byla realizována připojením přípravku k **analyzátoru CANalyzer** od firmy Vector [7]. Analyzátor CANalyzer byl schopen zasílat a přijímat rámce sběrnice CAN včetně změny rychlosti sběrnice CAN.

Testování pomocí analyzátoru proběhlo opět bez potíží. Přípravek správně přijímal i odesílal jednotlivé CAN rámce při různém nastavení rychlosti sběrnice CAN.

5.2 Testování konzolovou aplikací

Aplikace „CAN Explorer“ podporuje jen omezené množství příkazů protokolu USB. Zároveň v aplikace nelze nastavit konkrétní čas odeslání rámce CAN na sběrnici.

Z tohoto důvodu bylo nutné vytvořit vlastní konzolovou aplikaci, která bude schopna testovat zmíněné nedostatky aplikace „CAN Explorer“.

Testovací konzolová aplikace umožňuje spouštět několik různých typů testů, které odstraňují omezené možnosti aplikace „CAN Explorer“:

- Test konfigurace a čtení časovače rámců sběrnice CAN.
- Test vyčítání a zápisu registrů periferie CAN.
- Test časování rámců sběrnice CAN.
- Zaslání příkazu pro přechod do režimu DFU.

Konzolová aplikace pro testování zařízení je součástí přiloženého CD diplomové práce. Po spuštění této aplikace uživatel zvolí typ testu. Poté proběhne samotné testování, po kterém je zobrazena informace o průběhu a úspěchu či neúspěchu provedeného testu.

Konzolová aplikace obsahuje kromě ucelených testů jednoduchý systém pro zasílání jednotlivých příkazů do zařízení.

5.3 Testování DFU

Pro testování funkce aktualizace firmwaru bylo zapotřebí implementovat několik různých firmwarů standardního režimu zařízení. Jednotlivé firmwary byly zkompilovány do samostatných binárních souborů.

Binární soubory reprezentující různé standardní režimy byly převedeny na obrazy DFU [1] a postupně nahrávány prostřednictvím mechanismu DFU do hardwarového přípravku [2]. Vždy po aktualizaci hardwarového přípravku novým obrazem DFU (novým firmwarem standardního režimu) se testovalo, zda došlo ke změně funkcionality přípravku, který odpovídal nově nahranému firmwaru.

Při testování funkce DFU nedošlo k žádným problémům a hardwarový přípravek se vždy choval podle očekávání.

[1] Převod binárního souboru na obraz DFU popisuje kapitola 4.8.1 „Generování obrazu DFU“.
[2] Způsob přenosu obrazu DFU do hardwarového přípravku obsahuje kapitola 4.8.2 „Aktualizace firmwaru“.

Kapitola 6

Závěr

Prvním krokem diplomové práce bylo studium principu fungování sběrnic USB a CAN a použitého hardwarového přípravku. Poté jsem provedl rozbor fungování periferií hardwarového prostředku, které bylo potřeba použít v rámci diplomové práce. Dále bylo nutné podrobně nastudovat předchozí práce, na které má práce navazuje, a identifikovat problémy předchozí implementace vyvíjeného firmwaru.

Na základě získaných poznatků jsem navrhl hierarchický model nového firmwaru, který respektuje požadavky zadání diplomové práce na modularitu a přenositelnost knihoven pro periferie USB a CAN.

Navržený hierarchický model definuje jednoduché programové rozhraní mezi použitým hardwarem a aplikační vrstvou firmwaru. Díky tomu lze poměrně snadným způsobem implementovat různé aplikační vrstvy hardwarového přípravku využívající periferie USB a CAN, které realizují jeho různé funkce.

Jedním z bodů zadání diplomové práce je implementace existujícího protokolu ovladače převodníku USB/CAN pro operační systém Microsoft Windows XP. Tento protokol je v práci úspěšně implementován samostatnou aplikační vrstvou. Následně i testován testovacím módem periferie CAN, reálným připojením na sběrnici CAN a samostatnou aplikací vytvořenou v rámci diplomové práce přímo pro potřeby testování dané aplikační vrstvy.

Výsledná diplomová práce obsahuje mechanismus pro aktualizaci firmwaru hardwarového přípravku prostřednictvím USB. Společně s možností jednoduché implementace nových aplikačních vrstev získává výsledná práce silný nástroj pro snadnou změnu funkce hardwarového přípravku.

Součástí diplomové práce je programátorská dokumentace vygenerovaná do formátu HTML, usnadňující orientaci ve funkcích jednotlivých modulů systému.

Domnívám se, že všechny požadavky uvedené v zadání diplomové práce byly splněny, navíc byl v rámci diplomové práce implementován mechanismus pro aktualizaci firmwaru prostřednictvím USB.

Další práce mohou na mou diplomovou práci navazovat implementací nových aplikačních vrstev, implementací nových modulů, které zpřístupní aplikační vrstvě další periferie, nebo mohou implementovat moduly pro odlišný hardware již používaných periferií.

Literatura

[1] Apache Subversion - hlavní stránka.
http://subversion.apache.org/, stav z 22. 4. 2012.

[2] USB Device Class Specification for Device Firmware Upgrade, Version 1.1, 2004.

[3] Universal Serial Bus Specification, Revision 2.0. Technical report, Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V., 2000.

[4] Virtuální stroj - wikipedia.
http://cs.wikipedia.org/wiki/Virtu%C3%A1ln%C3%AD_stroj, stav z 22. 4. 2012.

[5] STMicroelectronics - Home Page, .
http://www.st.com/internet/com/home/home.jsp, stav z 6. 5. 2012.

[6] CAN bus Bit-Timing, .
http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/bit-timing.php?navanchor=3010534, stav z 1. 5. 2012.

[7] CANalyzer - Home Page, .
http://www.vector.com/vi_canalyzer_en.html, stav z 7. 5. 2012.

[8] IEEE 1149.1, .
http://standards.ieee.org/findstds/standard/1149.1-1990.html, stav z 6. 5. 2012.

[9] Managing Devices - USB Topology, 2003.
http://technet.microsoft.com/en-us/library/bb457107.aspx, stav z 1. 5. 2012.

[10] USB in a NutShell - Making sense of the USB standard, .
http://www.beyondlogic.org/usbnutshell/usb1.shtml, stav z 1. 5. 2012.

[11] Bc. Dušan Talíř. Převodník rozhraní USB – CAN. Master's thesis, ČVUT-FEL, 2006.

[12] Bc. Jan Kravar. Programové vybavení pro diagnostiku automobilových řídicích jednotek. Master's thesis, ČVUT-FEL, 2008.

[13] Bradley Mitchell. OSI Model - Open Systems Interconnection model.
http://compnetworking.about.com/cs/designosimodel/g/bldef_osi.htm, stav z 5. 5. 2012.

[14] doc. Ing. Jiří Novák, Ph.D. Prezentace: CAN bus a jeho aplikace ve vozidlech, .

[15] doc. Ing. Jiří Novák, Ph.D. Controller Area Network (CAN) - stručný popis, .

[16] doc. Ing. Miroslav Kubíček, CSc. Univerzální sériové rozhraní USB. http://www.odbornecasopisy.cz/index.php?id_document=27823, stav z 1. 5. 2012.

[17] STMicroelectronics. STR71x firmware library, 2007.

[18] STMicroelectronics. STR71x Microcontroller Reference Manual, Rev. 7, 2005.

[19] STMicroelectronics. STR7/STR9 USB developer kit, 2008.

[20] Václav Novák. Programové vybavení převodníku USB-CAN, 2006.

[21] WILLIAMS, R. N. A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS. 1993. http://www.cse.sc.edu/~jimdavis/Courses/2004-Fall%20CSCE%20491/crc-Ross.pdf, stav z 5. 5. 2012.

Příloha A

Seznam použitých zkratek

ACK Acknowledge

APB Advanced Periphetal Bus

ARM Advanced RISC Machine

BIN Binary

CAN Controller Area Network

CRC Cyclic redundancy check

CSMA/CR Carrier Sense Multiple Access with Collision Resolution

DFU Direct Firmware Update

GUI Graphical user interface

HEX Hexadecimal

HTML HyperText Markup Language

ID Identifier

IDE Integrated development environment

IDL Interface description language

IEEE Institute of Electrical and Electronics Engineers

JTAG Joint Test Action Group

LEC Last Error Code

LLC Logical Link Control

MAC Medium Access Control

NRZI Non Return to Zero Inverted

PC Personal computer

PDF Portable Document Format

PHP Hypertext Preprocessor

RAM Random access memory

RISC Reduced Instruction Set Computer

RTF Rich Text Format

RTR Remote Request

SIE Serial Interface Engine

SOF Start Of Frame

SVN Subversion

UART Universal asynchronous receiver/transmitter

USB Universal Serial Bus

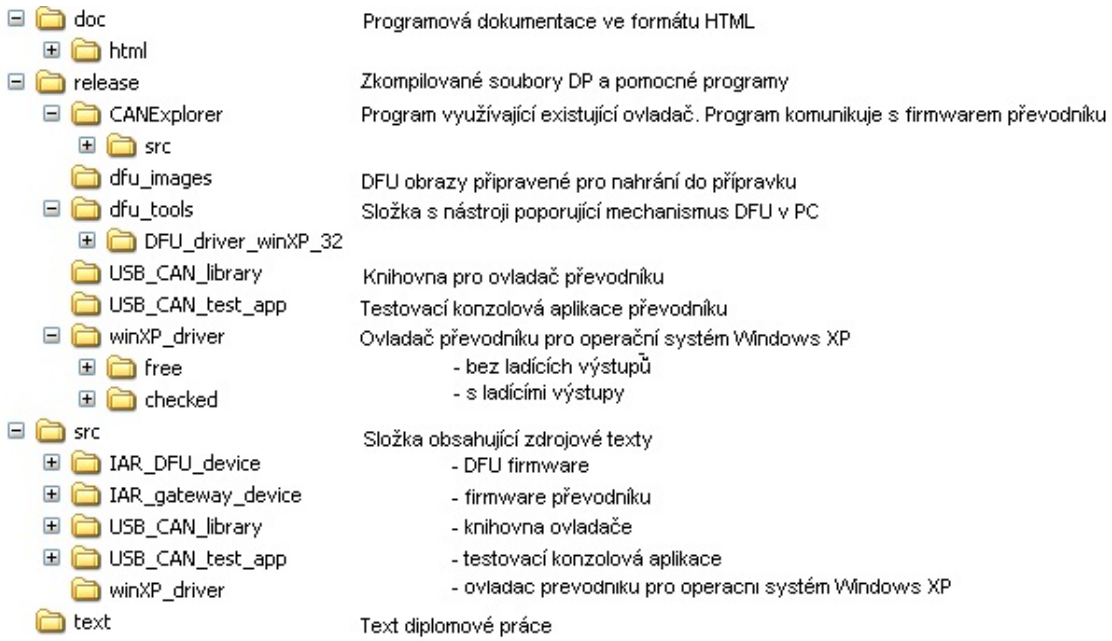
VHDL VHSIC hardware description language

WDK Windows Driver Kit

XML Extensible Markup Language

Příloha B

Obsah přiloženého CD



Obrázek B.1: Obsah přiloženého CD