ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ v PRAZE
Fakulta elektrotechnická
Katedra řídící techniky

Řadič sběrnice CAN připojený k PC přes sběrnici USB

Bakalářská práce

Vedoucí práce: ing. Pavel Píša
Autor práce: Jan Kříž

červenec 2008

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, atd.) uvedené v přiloženém seznamu.

V Praze, dne 11. 7. 2008 .....

podpis

Poděkování

Rád bych poděkoval vedoucímu této práce ing. Pavlu Píšovi, který mi poskytl veškeré potřebné technické vybavení, měl se mnou trpělivost a dával mi cenné rady jak při tvorbě programového vybavení, tak i při psaní této práce. V neposlední řadě chci poděkovat mé rodině a přátelům za podporu při práci.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jan Kříž**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Řadič sběrnice CAN připojený k PC přes sběrnici USB**

Pokyny pro vypracování:

1. Seznamte se s vlastnostmi a funkcí sběrnice CAN. Připravte krátký přehled vhodných mikroprocesorových obvodů pro realizaci převodníku USB/CAN s dostatečně vekou vyrovnávací pamětí. Soustřeďte se především na levnější a menší obvody s architekturou ARM, které mají dostatečné množství paměti RAM a FLASH.
2. Seznamte se s návrhem hardware realizovaným společností PiKRON, který je založený na mikrokontroléru LPC2148 v kombinaci s řadičem CAN SJA1000.
3. Navrhněte firmware převodníku a adaptaci driveru pro počítač PC. Zhodnoťte řešení a případně navrhněte další alternativy.

Seznam odborné literatury:

lLPC214x User Manual, LPC214x revision, Philips Electronics N.V., 2005
lWebové stránky projektu OCERA http://www.ocera.org/
lWebové stránky obdobného projektu pro Windows http://www.pp2can.wz.cz/
lCorbet Corbet, Alessandro Rubini, Greg Kroah-Hartman, Linux Device Drivers, 3rd Edition, O'Reilly Media, Inc., 2005, ISBN-10: 0596005903, http://lwn.net/Kernel/LDD3/

Vedoucí: Ing. Pavel Píša

Platnost zadání: do konce zimního semestru 2008/2009

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 29. 2. 2008

Abstrakt

Cílem práce je seznámit se s vlastnostmi a funkcí sběrnice CAN, připravit přehled mikroprocesorových obvodů, především s architekturou ARM, vhodných pro realizaci převodníku USB/CAN a dále se seznámit s hardware založeným na mikrokontroléru LPC2148 v kombinaci s řadičem sběrnice CAN SJA1000 navrženým společností PiKRON.

Pro dodaný hardware navrhnout firmware převodníku a adaptaci ovladače pro počítač PC. Realizovaný převodník vyzkoušet a zhodnotit řešení.

Klíčová slova: CAN, USB, Linux, převodník.

Abstract

The purpose of this work is to learn about principles and function of the CAN bus, create a list of suitable microprocessor circuits, in particular with the ARM architecture, and to get to know of a hardware designed by PiKRON company with an LPC2148 microprocessor and an SJA1000 CAN bus driver.

The main part of the work consists of a firmware design for the provided hardware and a driver adaptation for a PC computer to create a working USB/CAN converter. The final part discusses the testing phase and results.

Keywords: CAN, USB, Linux, converter.

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Sběrnice CAN[1] je v průmyslu velmi dobře rozšířená. Má ochranné mechanizmy přenosu dat a umožňuje řízení v reálném čase. Používá krátké zprávy přenášené nejčastěji po dvouvodičové sběrnici, jejíž nejvyšší rychlost je 1 Mbit/s.

Ve vyšších vrstvách řízení se v průmyslu vzhledem k ceně často používají i počítače typu PC. Stolní a především přenosné počítače (laptopy) se používají k monitorování, konfiguracím a diagnostice probíhajících průmyslových procesů. Vkládání specializovaných počítačových karet pro komunikaci s průmyslovými sběrnicemi však u nich bývá problematické. Vhodným řešením se jeví využití rozhraní USB, kterým je vybaven každý moderní laptop nebo PC, pro připojení převodníku sběrnice CAN.

Cílem práce je po úvodním rozboru hardwarových možností navrhnout firmware převodníku USB/CAN pro modul s mikroprocesorem s architekturou ARM propojeným s řadičem sběrnice CAN a integrovat podporu modulu do ovladače sběrnice CAN pro operační systém Linux.

[1] Controller Area Network

2 Sběrnice CAN

Datová komunikační síť CAN je sběrnice typu *multi-master*[2]. Byla navržena koncem 80. let firmou Robert Bosch GmbH pro aplikaci v automobilech. Původním záměrem byla úspora kabelového vedení při vysoké spolehlivosti a fungování v reálném čase[3], proto také první verze používala modifikovanou sběrnici RS 485. Díky relativně vysoké rychlosti komunikace a vysoké odolnosti proti rušení se tato sběrnice prosadila v automobilovém a později i v dalším průmyslu. To s sebou přineslo snížení cen zařízení schopných komunikovat na sběrnici CAN a podpořilo rozšíření i do dalších aplikací.

Sběrnice CAN je chráněna patenty společnosti Robert Bosch GmbH a pro její používání je nutno získat licenci. Licenční poplatky jsou většinou přenášeny na spotřebitele přímo v ceně integrovaných obvodů.

Sběrnice CAN ve verzi 2.0 byla uvedena na trh v roce 1991. Postupem doby se modifikovala do dvou vzájemně kompatibilních systémů CAN 2.0A a CAN 2.0B. V roce 1993 byla sběrnice standardizována mezinárodním standardem ISO 11898, v České republice pak v normě ČSN EN 50325.

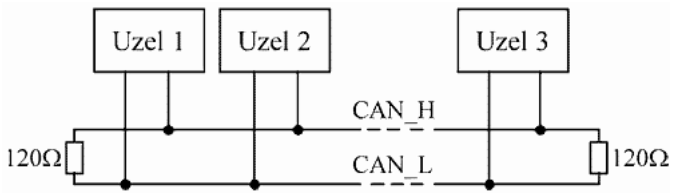
2.1 Základní vlastnosti

- Sběrnice dosahuje relativně vysoké přenosové rychlosti (až 1 Mbit/s).
- Sběrnice je typu *multi-master*, není tedy závislá na žádném arbitrujícím uzlu, komunikace je decentralizovaná.
- Priorita přenášených zpráv je rozlišena identifikátorem v hlavičce každé zprávy, při současném posílání dvou zpráv *vyhraje* uzel přenášející zprávu s *dominantnějším* identifikátorem.
- Ochranné mechanizmy – počítadla chyb umožňují odpojení chybného uzlu pro bezpečný běh ostatních, chybové zprávy propagují chybový stav do všech uzlů.
- Ochrana typu CSMA/CA[4] proti kolizím v komunikaci mezi uzly.

2.2 Fyzická vrstva

Typické zapojení sběrnice CAN je naznačeno v obrázku **2.1**. Její specifikace není vázána na určité přenosové médium, komunikace může probíhat napří-

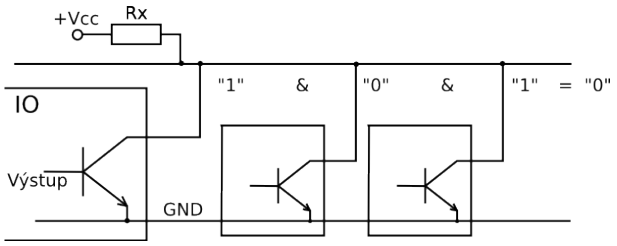
[2] Každý uzel může řídit komunikaci
[3] pokud se uvažuje reálný čas v řádu milisekund
[4] Carrier Sense Multiple Access With Collision Avoidance



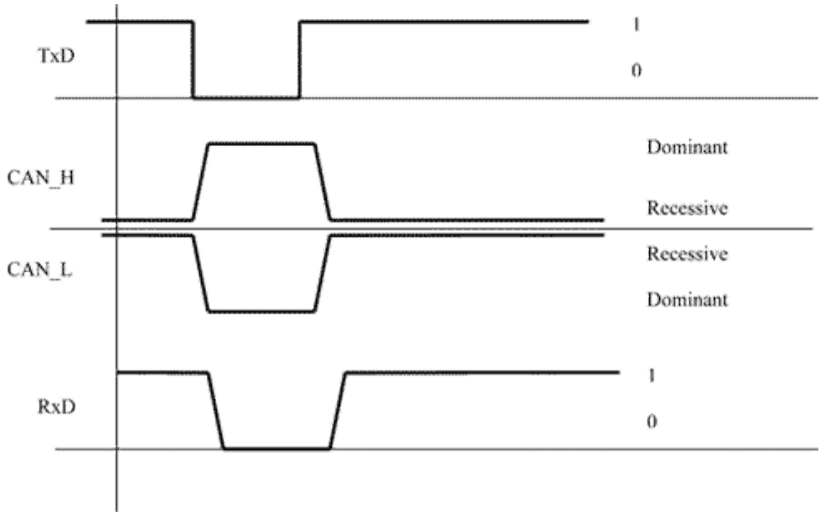
Obrázek **2.1**: Příklad zapojení sběrnice CAN [2]

klad optickou cestou. Nejběžnější je však přenos po metalickém vedení. Normou ISO 11898-2 je definováno vyvážené dvouvodičové diferenciální vedení signálu. Je obvyklé v automobilových sběrnicích a je také použito v převodníku společnosti PiKRON, který je popsán níže v této práci. Vyvážené diferenciální vedení znamená dva vodiče, které mají stejné vlastnosti, ale v jednom parametru se doplňují (parametry jsou protichůdné), například směr toku elektrického proudu, nebo napěťové úrovně. Další informace je možné získat v dokumentech [8, 9].

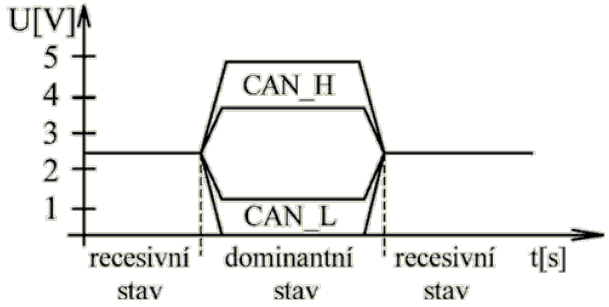
Na sběrnici se přenáší dva stavy – recesivní a dominantní. V bitových úrovních je dominantní 0 a recesivní 1. Při použití budičů sběrnice s otevřeným kolektorem (obrázek **2.2**) v jednotlivých uzlech potom vyslaný dominantní bit překryje všechny recesivní. Pokud je vysílán recesivní bit, ale detekován dominantní, pak nastala kolize nebo chyba v přenosu. Toho je využíváno při posílání zpráv na sběrnici. Uzly monitorují provoz a pokud není vysílána zpráva, začnou vysílat. Záhlaví zpráv jsou stejná a vysílané bity se mohou lišit až v identifikátoru zprávy. Uzel při detekci kolize v identifikátoru, která nastane jedině, pokud je jiným uzlem ve stejnou chvíli vysílán identifikátor vyšší důležitosti, skončí vysílání a zkouší vysílat znovu po skončení komunikace na sběrnici. Tyto postupy zajišťují ochranu proti kolizím (CSMA/CA).



Obrázek **2.2**: Příklad zapojení s otevřeným kolektorem



Obrázek **2.3**: Úrovně signálu na sběrnici CAN [3]



Obrázek **2.4**: Toleranční pásmo napěťových úrovní [2]

Je-li sběrnice v recesivním stavu, pak mezi oběma vodiči je nulové napětí. V dominantním stavu je na vodiči CAN_H napětí v rozsahu 3,5 až 5 V a na vodiči CAN_L napětí v rozsahu 0 až 1,5 V. Časový průběh elektrického napětí na vedení je uveden v obrázku **2.3**. Napěťová toleranční pásma pro vodiče CAN_H a CAN_L jsou uvedena v obrázku **2.4**. Vedení musí být na obou koncích zakončeno rezistory s odporem o velikost 120 Ω zamezujícími odrazům signálu.

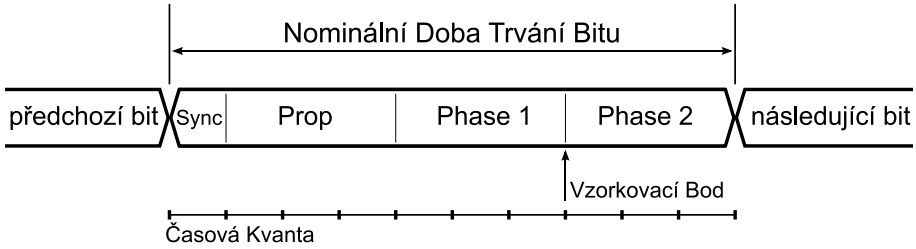
Přenosová rychlost je omezena délkou spoje a není závislá na počtu připojených uzlů. Orientační údaje rychlosti sběrnice jsou uvedeny v tabulce **2.1**.

Sběrnice nemá žádný synchronizační vodič, proto je synchronizace všech zařízení řešena během komunikace. K synchronizaci dochází během přenosu

Tabulka **2.1**: Rychlost sběrnice CAN v závislosti na délce spoje

Délka sběrnice [m]	Komunikační rychlost [kbit/s]
<40	1000
≈ 130	500
≈ 560	125
≈ 3300	20

každého bitu. Každý přenášený bit zprávy se skládá z několika časových segmentů (obrázek **2.5**).



Obrázek **2.5**: Segmentace jednotlivých bitů CANové zprávy [4]

- SYNC_SEG – Začátek přenášeného bitu.
- PROP_SEG – Segment časového zpoždění z důvodu propagace synchronizačního bitu do všech uzlů na sběrnici.
- PHASE_SEG1 a PHASE_SEG2 – Délky těchto segmentů si nastavuje každý kontrolér sám v závislosti na aktuálním stavu sběrnice. Bod sejmutí hodnoty bitu ze sběrnice je pak mezi oběma segmenty.

Aby se zamezilo chybám v komunikaci, je při výskytu 5 stejných bitů po sobě v průběhu vysílání zprávy automaticky vložen bit opačné polarity, který je na protějších uzlech automaticky odstraněn, takže se ve zprávě neprojeví. Této technice se říká *bit stuffing*. Pokud je detekována šestice stejných bitů, pak je považována za chybu.

2.3 Spojová vrstva

Pro komunikaci na sběrnici CAN verze 2.0 jsou definovány dva typy přenosových rámců: CAN 2.0A s identifikátorem standardní délky a CAN 2.0B s rozšířeným identifikátorem. Jediným rozdílem je tedy délka identifikátoru 11 bitů pro CAN 2.0A a 29 bitů pro CAN 2.0B. Při komunikaci je délka identifikátoru určena bitem IDE, který je dominantní pro standardní identifikátor a recesivní pro rozšířený identifikátor.

2.3.1 Datový rámec (*Data frame*)

Standardní rámec pro přenosy dat. Rámec se skládá z těchto částí

- Start of Frame – úvodní jednobitové pole s dominantní hodnotou.
- Arbitration Field – pole sestávající se z identifikátoru zprávy a bitu RTR (Remote Transmission Request), který určuje, zda se jedná o datový rámec (data frame) nebo žádost o vysílání (remote frame).
- Control Field – řídící pole, obsahující bit IDE (identifier extension) pro rozlišení základního a rozšířeného formátu, rezervní bit a 4 bity DLC (Data Length) určující počet datových bytů (0 až 8 bytů).
- Data Field – datové pole o velikosti 0 až 8 bytů.
- CRC Field (Cyclic Redundancy Code) – 15 bitů kontrolního kódu vyjadřujícího předešlá pole. Pole je ohraničeno recesivním bitem ERC (END OF CRC).
- Acknowledge Field – potvrzující pole, které sestává z bitů ACK Slot a ACK Delimiter. Vysílač vysílá bit ACK Slot jako recesivní. Pokud alespoň jeden uzel přijal zprávu bez chyby, přepíše tento bit na dominantní, čímž oznámí vysílači potvrzení příjmu. ACK Delimiter je recesivní bit, takže ACK Slot je ohraničen dvěma recesivními bity.
- End of Frame – konec rámce se skládá z nejméně sedmi recesivních bitů, za nimiž následují nejméně 3 bity pro uklidnění všech vysílačů. V této době mohou přijímací uzly informovat vysílací uzel o chybách přenosu.
- Intermission Field + Bus Idle – mezilehlé pole + uklidnění sběrnice – 3 bity oddělující jednotlivé zprávy.

K obsahu identifikátoru je dáno omezení, že počáteční sedmice bitů nesmí být celá recesivní (tedy identifikátor 1111111xxxx je neplatný).

Při vysílání identifikátoru uzel monitoruje stav na sběrnici. Pokud vyslal recesivní bit, ale na sběrnici je bit dominantní, znamená to, že ve stejnou dobu jiný uzel vysílá důležitější zprávu, a okamžitě přestává vysílat. Tak se na sběrnici dostane nejdříve zpráva s nejvyšší prioritou. Tím, že je jedna ze zpráv vyslána, je zabezpečeno, že při nárůstu zatížení sběrnice nedochází ke snížení přenosového výkonu sítě.

Počet datových bytů je omezen na maximálně 8. Tato poměrně malá délka informace vychází z původního záměru CAN, tj. především zabezpečení rychlého přenosu zpráv s vysokou prioritou. Delší bloky dat je nutno segmentovat v aplikační úrovni. Všechna data na sběrnici jsou dostupná všem uzlům současně.

Grafické znázornění rámců obou typů je uvedeno v příloze A.

2.3.2 Žádost o rámec (*Remote frame*)

Jde o krátkou zprávu, která je požadavkem na odeslání zprávy vzdáleným uzlem. Neobsahuje tedy žádná data a má bit RTR recesivní, jinak se neliší od datového rámce. Navrácený datový rámec by měl obsahovat stejný identifikátor.

Tímto způsobem lze realizovat řízení typu *master-slave* oproti standardnímu *multi-master* řízení.

2.3.3 Chybový rámec (*Error frame*)

Používá se v případě detekce chyby na sběrnici, například pokud během vysílání zprávy je mimo oblast identifikátoru vysílán recesivní bit, ale detekován je dominantní.

Skládá se ze tří částí:

- chybové návěstí (Error Flag) - skládá se ze šesti dominantních bitů, čímž se poruší integrita zprávy a ostatní uzly začnou také vysílat chybový rámec,
- čekání na odezvu všech uzlů - má maximálně 6 bitů,
- konec bloku (Error Delimiter) - obsahuje 8 recesivních bitů.

2.3.4 Zpožďovací rámec (*Overload frame*)

Má podobnou strukturu jako chybový rámec. Vysílá se při dvou událostech:

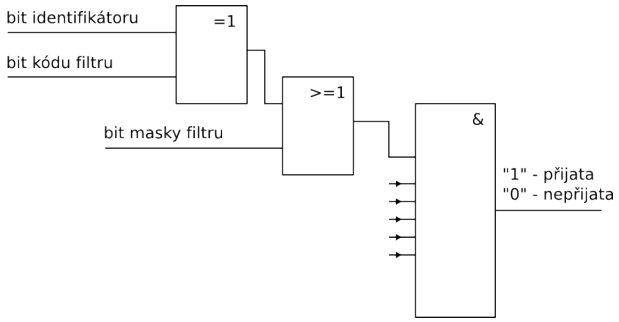
- V závislosti na stavu přijímače, kdy potřebuje nějaký čas na zpracování předchozí zprávy. Zpožďovací rámec pak může být zahájen pouze v okamžiku 1. bitu doby mezi rámci.
- Při detekci dominantního bitu v době mezi dvěma rámci, rámec musí začít v náledujícím bitu po detekci.

2.4 Filtrování zpráv

Identifikátory zpráv slouží nejen pro rozhodování o prioritě vysílání zpráv na sběrnici, ale především určují typ zprávy v dané aplikaci a jsou obvykle používány pro adresaci cílových uzlů. Po sběrnici CAN často komunikuje řada různorodých zařízení. To s sebou nese i velké množství zpráv s různými identifikátory.

Zpracovávání všech zpráv přicházejících po sběrnici firmwarem procesoru je velmi často neefektivní, především pokud daná aplikace používá pouze omezenou sadu identifikátorů a ostatní zprávy jsou zahazovány. Z tohoto důvodu řadiče sběrnice CAN implementují filtry zpráv přicházejících ze sběrnice. Tím je umožněno snížit nároky na výkon procesoru, ke kterému je řadič připojen.

Filtry zpráv se skládají ze dvou částí – z kódu a masky. Maska slouží pro určení, které bity identifikátoru se budou porovnávat. V těch bitech, kde je maska nulová, záleží na obsahu filtrovacího kódu. Pokud jsou veškeré srovnávané bity identifikátoru shodné s kódem filtru, je zpráva přijata. Postup je znázorněn na obrázku **2.6**.



Obrázek **2.6**: Filtrování zpráv na sběrnici CAN

3 Sběrnice USB

USB je sériová datová sběrnice, která umožňuje propojení hostitelského zařízení (převážně osobního počítače) a periferních zařízení. Sběrnice se prosadila díky jednoduchému způsobu připojení, možnosti připojování a odpojování za běhu hostitelského stroje, poměrně vysoké rychlosti toku dat (ve verzi 1.1 byla nejvyšší rychlost stanovena na 12 Mbit/s, ve verzi 2.0 rychlost 480 Mbit/s), možnosti napájení zařízení přímo po sběrnici a levnou implementací do již existujících zařízení.

Velkou výhodou sběrnice je podpora typů (tříd) zařízení (přehled uveden v [6]), takže například všechny skenery mají stejný způsob komunikace s hostitelským zařízením a není potřeba zvláštních ovladačů. Zařízení nezařaditelná do těchto typů, jako je níže popsaný převodník, potřebují vytvořit vlastní ovladač pro daný operační systém.

3.1 Historie sběrnice

Sběrnice USB byla uvedena na trh ve verzi 1.0 v roce 1995. Po prvních zkušenostech byla v září 1998 uvedena specifikace verze 1.1, která mj. opravovala chyby v kompatibilitě zařízení vycházejících z verze 1.0. Byla propagována velkými společnostmi jako Intel (*UHCI* a *open software stack*), Microsoft (*Windows software stack*), Philips (*Hub*, *USB-Audio*) a US Robotics. Specifikace ve verzi 2.0 byla uvedena v dubnu 2000 a standardizována v roce 2001 organizací USB-IF (*USB Implementers Forum*). Masového rozšíření se sběrnice dočkala v revizi 1.1.

Počítač Apple „Bondi blue“ iMac G3, uvedený na trh 6. května 1998 byl prvním, který podporoval USB jako standard včetně připojení klávesnice a myši.

Sběrnice podporuje zařízení, která nejsou datově náročná (např. kávesnice nebo myš), ale také náročnější zařízení (skenery apod.), proto specifikace verze 1.1 rozlišuje mezi *low-speed devices* (LS) pracujících na rychlosti 1,5 Mbit/s a *full-speed devices* (FS) fungujících s plnou rychlostí 12 Mbit/s. Specifikace 2.0, která je zpětně kompatibilní s verzí 1.1, rozšiřuje rychlosti komunikace o *high-speed devices* (HS) pracujících s rychlostí 480 Mbit/s. Další podrobnosti jsou uvedeny v [5] v kapitole 2.1 a v [6].

3.2 Fyzická vrstva

Komunikace probíhá po čtyřvodičovém kabelu, kde dva vodiče jsou datové a zbylé dva napájecí. Napájecí napětí je 5 V se stejnosměrným proudem. Fy-

zickou délku sběrnice lze zvětšit použitím až 5 úrovní rozbočovačů (1. vrstvou je kořenový rozbočovač v hostiteli), kdy každý s sebou nese určitý povolený úbytek napětí (přesné hodnoty jsou popsány v [5]), proto musí připojená zařízení být schopna pracovat již od 4,4 V. Délka kabelu je omezena na 5 m, ale díky možnosti použití rozbočovačů lze použít zařízení až do vzdálenosti 30 m.

Proudová zatížitelnost sběrnice se liší podle stavu zařízení. V režimu spánku smí zařízení čerpat max. 0,5 mA, zařízení s malým odběrem smí čerpat nejvýše 100 mA. Zařízení *s velkým odběrem proudu* nesmí překročit odběr 500 mA a musí být schopna pracovat od 4,75 V.

Fyzická topologie sběrnice je stromová, k hostitelskému rozbočovači mohou být připojena zařízení, nebo pasivní či aktivní rozbočovače. Maximální počet zařízení na sběrnici je 127. Sběrnice je arbitrována hostitelem, který kontaktuje jednotlivé uzly s požadavky na příjem nebo odesílání dat. Přes tyto vlastnosti umí USB subsystém navíc zajistit konstantní datový tok pro přenos video či audio streamů.

Podrobnější informace lze nalézt v [5].

3.3 Spojová vrstva

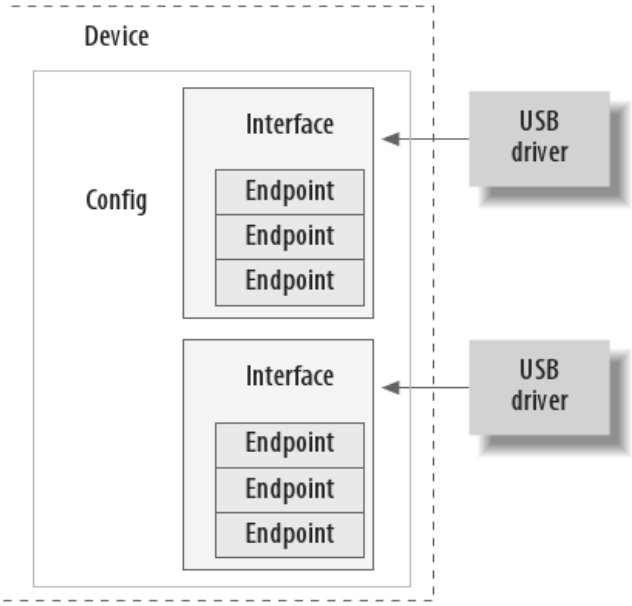
Každé připojené zařízení může být schopno zajistit i více různorodých funkcí. Z tohoto důvodu má každé zařízení hierachii popsanou v bodech 3.3.1, 3.3.2 a 3.3.3 a znázorněnou v obrázku **3.1**.

3.3.1 Konfigurace (*Configuration*s)

Každé zařízení může obsahovat více konfigurací, mezi kterými lze přepínat. Efekt přepínání konfigurací může být různý pro každé zařízení. Může jít například o přepínání funkčních režimů zařízení (například režim standardní funkce a režim přeprogramování zařízení). Každá konfigurace může sdružovat jedno nebo více rozhraní.

3.3.2 Rozhraní (Interfaces)

Jedno zařízení připojené na USB může mít více funkčností a typů (tříd), tedy například multifunkční tiskárna se může chovat jako oddělená tiskárna, skener a fax, každý se svou třídou zařízení a tedy bez potřeby zvláštních ovladačů. Každé rozhraní může obsahovat jeden nebo více koncových bodů.



Obrázek **3.1**: Hierarchie USB zařízení [1]

3.3.3 Koncové body (*Endpoint*s)

Endpointy, dále jen EP, jsou jednoznačně identifikovatelné koncové body komunikace s hostitelským zařízením. Každé zařízení na sběrnici musí mít nejméně jeden (řídící) EP, ale mohou jich mít více. Pro LS zařízení je počet EP omezen na 3 brány, pro FS je horní hranice 16 EP.

Všechna zařízení musí povinně obsahovat *endpoint 0*, který je řídící (*Control EP*). Ostatní EP jsou volitelné. Mohou mít jeden ze čtyř typů.

Řídící EP (*Control EP*). Slouží k identifikaci zařízení a získání informací o konfiguracích, rozhraních a dalších EP, které může zařízení obsahovat. Tomuto procesu se říká *enumerace* zařízení a probíhá po připojení zařízení ke sběrnici nebo resetu zařízení.

Řídící EP dále slouží během provozu k přepínání konfigurací a vyřizování uživatelských požadavků (*vendor requests*).

Obsluha přerušení (*Interrupt EP*). Přenáší malé objemy dat s konstantní frekvencí dotazů od hostitele. Tato metoda je používána například u počítačových klávesnic a myší. Koncové body tohoto typu však nejsou ur-

čeny pro přenos větších objemů dat. Mají specifikací zajištěnou dostatečnou šířku pásma, proto by při nadměrných objemech dat blokovaly komunikaci s dalšími zařízeními.

Standardní EP (*Bulk EP*). Slouží pro přenos velkých objemů dat. Jsou běžně používané v zařízeních, kde je třeba přenášet data bezztrátově. Přenosy standardními koncovými body nemají zaručeno, že dorazí do cíle ve specifikované době. Pokud je objem přenášených dat větší, než se vejde do časového rámce, jsou data rozdělena na více paketů. Koncové body tohoto typu se používají v tiskárnách, úložných zařízeních, síťových zařízeních a dalších.

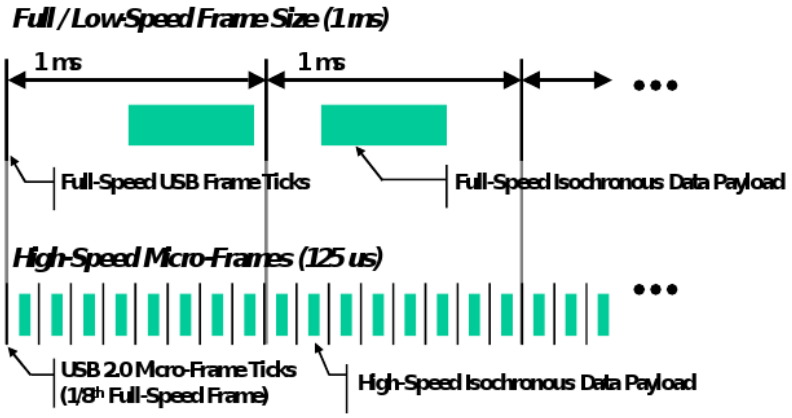
Izochronní EP (*Isochronous EP*). Stejně jako standardní EP přenáší izochronní EP velké objemy dat, ale důraz je kladen na konstantní rychlost datového toku. Pokud nemohou být data přenesena v požadované době, jsou zahozena. Tento typ EP je vhodný pro zařízení pracující s tokem zvukových nebo obrazových dat.

3.3.4 Přenosy

Specifikace USB definuje časové rámce (*frame*) znázorněné na obrázku **3.2** pro komunikaci se všemi zařízeními. Tyto rámce trvají $1,00\,ms \pm 0,0005\,ms$ pro FS a LS[5] a $125\,\mu s \pm 0,0625\,\mu s$ pro *HS*[6]. Každý rámec začíná paketem Start of Frame (SOF), kterým je možno synchronizovat časovač zařízení, potom již probíhá komunikace se zařízeními. Komunikace probíhá prostřednictvím paketů. Jejich typy a složení jsou popsány v [5], kapitole 2.2.2.1.

Požadavek na čtení nebo zápis z/do zařízení proběhne nejvýše jednou za jeden rámec, takže není možné dosáhnout kratší latence než 1 ms pro FS a LS a 0,125 ms pro HS zařízení.

[5] *full-speed* a *low-speed* (USB 1.1)
[6] *high-speed* (USB 2.0)



Obrázek **3.2**: Časové rámce USB komunikace [7]

4 Architektura ARM

ARM (Advanced RISC Machine, dříve Acorn RISC Machine) je 32-bitová architektura využívající omezeného instrukčního souboru (RISC), která je často používána v malých elektronických zařízeních (embedded devices).

V dnešní době mají procesory s jádry ARM podíl zhruba 75 % na trhu všech 32-bitových RISC procesorů. Díky nízké spotřebě najdeme tyto procesory nejčastěji v bateriových mobilních přístrojích jakými jsou mobilních telefony (Nokia, Sony Ericsson, Samsung), PDA zařízení, kapesní přehrávače (Apple iPod a iPhone), GPS zařízení, fotoaparáty, ale také počítačové disky, síťové routery a dalších zařízení.

4.1 Stručná historie

ARM architektura začala být vytvářena v roce 1983 jako projekt společnosti Acorn Computers. Původním záměrem společnosti bylo vyrábět osobní počítače. Po několika vydaných modelech (Acorn Atom, BBC Micro, Acorn Electron), které obsahovaly 8-bitové procesory, vedení rozhodlo o přechodu na 16 nebo 32-bitovou architekturu. Poté, co Intel odmítl poskytnout architekturu jádra svého 286, se vývojáři rozhlíželi po dalších procesorech na trhu, mezi nimiž byly National 16032 a Motorola 68000, ale ty jim nevyhovovaly. Podle slov Steva Furbera, jednoho z vývojářů architektury, podávaly špatné výkony a měly příliš složitý instrukční soubor.

Bylo tedy rozhodnuto vyrobit vlastní jádro procesoru. Oproti vývojovým skupinám ostatních firem vyrábějících procesory tehdy nedostala skupina okolo Steva Furbera a Sophie Wilsonové žádné peníze na vývoj, ani žádné další lidi. Přes tyto překážky se návrh architektury podařil a 26. dubna 1985 již fungoval první procesor ARM. Vývoj trval 18 kalendářních měsíců.

V roce 1990 byla za podpory Acorn Computers, VLSI a Apple Computer založena společnost Advanced RISC Machines Ltd. Zvolená obchodní politika však byla jiná než u konkurence. Společnost nevyrábí čipy založené na své architektuře, ale licencuje svou technologii jako *intelektuální majetek* (Intellectual Property - IP). Myšlenkou je, aby zákazník přidal k zakoupenému jádru vhodné periferie, které dohromady vytvoří integrovaný obvod. Tak je možné vytvářet univerzální i specializované obvody, které mají nízkou cenu a přesto poměrně vysoký výkon. Proto je v dnešní době na trhu několik desítek společností vyrábějících své procesory založené na architektuře ARM, mezi kterými jsou i Intel, Renesas (dříve Hitachi a Mitsubishi) a Freescale (dříve Motorola).

Významnými kroky byly v posledních letech akvizice KEIL Software

(2005), hlavního vývojáře programových vývojových nástrojů (Software Development Kit - SDK) pro mikrokontroléry, akvizice společnosti Falanx (2006), zabývající se vývojem 3D grafických akcelerátorů, a společnosti SOISIC (2006), specializovanou na technologii výroby čipů metodou křemíku na izolantu (Silicon on Insulator - SOI).

Nejvýznamnější verze jádra byla ARM7TDMI, jíž se prodaly stovky milionů. Tato verze mimo jiné podporuje režim *Thumb*, zvyšující hustotu kódu v paměti. Jádro v tomto režimu vykonává 16-bitové instrukce. Mnoho instrukcí je přímo mapováno na původní 32-bitové instrukce a úspora prostoru tkví ve fixování operandů některých instrukcí a tím vynechání jejich některých méně používaných variant.

Z důvodu používání jader ARM ve složitějších zařízeních (mobilní telefony, přehrávače médií, apod.), kde jsou spouštěny aplikace v Javě, je v posledních verzích jádra implementována možnost vykonávání určitých příkazů bytového kódu (bytecode) Javy hardwarově a urychlit tím fungování aplikací. Toto rozšíření se jmenuje Jazelle DBX (Direct Bytecode eXecution – přímé vykonávání bytového kódu).

4.2 Vhodné procesory

K výrobě převodníku USB/CAN je potřeba použít procesory s dostatečným výkonem a výhodným poměrem cena/výkon. Nabízí se řada procesorů různých výrobců, každý s jinými vlastnostmi, mezi kterými je třeba vybrat. Následující výběry jsou omezeny pouze na zařízení s dostatečně velkou programovou pamětí vzhledem k velikosti firmware, tedy větší nebo rovnou 128 kB, obsahující řadič sběrnice USB. K dispozici jsou též zařízení schopná pracovat s externí programovou pamětí, ale pro dosažení minimálních rozměrů DPS[7] modulu je výběr směřován pouze na modely s pamětí integrovanou.

Řadič sběrnice CAN můžeme použít buďto vnější, pak je k dispozici například jeden ze standardních obvodů Intel i82527, Philips 82c200 nebo Philips SJA1000, nebo řadič integrovaný přímo v procesoru. Přehled procesorů s integrovaným řadičem sběrnice CAN je uveden v tabulkách **4.2** a **4.3**. Určitou výhodou při implementaci převodníku s vnějším řadičem CAN je připravenost kódu pro standardní typy čipů uvedené výše, neboť byly v programu pro mikroprocesor využity části ovladače LinCAN popsaného v kapitole 6.2, kde jsou implementovány. Nese to však s sebou i nutnost návrhu větší DPS. Pro oddělené řešení jsou parametry dostupných procesorů

[7] Deska plošných spojů

Název řadiče	Počet message objektů	Poznámka	Cena
Intel i82527	14	1 globální maska	10,40 $
Philips SJA1000T	1	režim PeliCAN umožňující použití rozšířené masky	5,43 $

Tabulka **4.1**: Přehled obvyklých externích řadičů sběrnice CAN

uvedeny v tabulce **4.4** a parametry externích řadičů v tabulce **4.1**.

U některých procesorů ve výše uvedených tabulkách jsou v poli cena uvedeny hodnoty N/A. To znamená, že obvod je dostupný, avšak cenu se z dostupných zdrojů nepodařilo získat.

Z uvedených tabulek vyplývá, že v dnešní době je na trhu již o mnoho více nových mikroprocesorů, které mají řadič sběrnice CAN integrovaný. Většina modelů obsahuje také mnoho dalších periferií, které však nebudou v převodníku využity. Nabízí se tedy možnost využít velké programové i operační paměti a vytvořit víceúčelový převodník. Jednou takovou možností, o níž je do budoucnosti uvažováno, je spojit funkci převodníku USB/CAN s převodníkem USB na sběrnici realizovanou v projektu uLan[8].

Rostoucí počet integrovaných řešení mikroprocesoru s řadičem sběrnice CAN s sebou pravděpodobně přinese potřebu návrhu nového převodníku, který tak bude mít nižší energetické nároky, bude zabírat menší plochu na desce plošných spojů a bude mít díky propojení řadiče s jádrem vnitřními sběrnicemi mikroprocesoru i jednodušší firmware.

[8] http://sourceforge.net/projects/ulan/

Název	Jádro	Paměť [kB]		Počet I/O	f_{MAX}	Pouzdro	Poznámka	Cena/ks
		Flash	RAM		[MHz]			
Atmel AT91SAM7A3	ARM7TDMI-S	256	32	62	60	LQFP100	USB2.0 FS	14,84 $/1
Atmel AT91SAM7X128	ARM7TDMI-S	128	32	62	55	LQFP100, TFBGA100	USB2.0 FS, Ethernet	10,11 $/1
Atmel AT91SAM7X256	ARM7TDMI-S	256	64	62	55	LQFP100, TFBGA100	USB2.0 FS, Ethernet	13,12 $/1
Atmel AT91SAM7X512	ARM7TDMI-S	512	128	62	55	LQFP100, TFBGA100	USB2.0 FS, Ethernet	18,21 $/1
NXP LPC2364	ARM7TDMI-S	128	34	70	72	LQFP100	USB2.0 FS, Ethernet	7,58 $/1
NXP LPC2366	ARM7TDMI-S	256	58	70	72	LQFP100	USB2.0 FS, Ethernet	9,02 $/1
NXP LPC2368	ARM7TDMI-S	512	58	70	72	LQFP100	USB2.0 FS, Ethernet	8,70 $/1
NXP LPC2378	ARM7TDMI-S	512	58	10	72	LQFP144	USB2.0 FS, Ethernet, řadič externí paměti	9,96 $/1
NXP LPC2387	ARM7TDMI-S	512	98	70	72	LQFP100	USB2.0 FS, Ethernet	12,21 $/1
NXP LPC2388	ARM7TDMI-S	512	98	10	72	LQFP144	USB2.0 FS Device/Host/OTG, Ethernet, řadič externí paměti	13,98 $/1
NXP LPC2458	ARM7TDMI-S	512	98	12	72	TFBG208	USB2.0 FS Device/Host/OTG, Ethernet, řadič externí paměti	11,88 $/1
NXP LPC2468	ARM7TDMI-S	512	98	16	72	LQFP208, TFBG208	USB2.0 FS Device/Host/OTG, Ethernet, řadič externí paměti	13,46 $/1
NXP LPC2478	ARM7TDMI-S	512	98	16	72	LQFP208, TFBG208	USB2.0 FS Device/Host/OTG, Ethernet, QVGA LCD řadič, řadič externí paměti	14,90 $/1
STM STM32F103CB	Cortex-M3	128	20	37	72	LQFP48	USB2.0 FS	8,12 $/1
STM STM32F103RB	Cortex-M3	128	20	51	72	LQFP64	USB2.0 FS	8,60 $/1
STM STM32F103RC	Cortex-M3	256	48	51	72	LQFP64	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103RD	Cortex-M3	384	64	51	72	LQFP64	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103RE	Cortex-M3	512	64	51	72	LQFP64	USB2.0 FS, řadič externí paměti	14,04 $/1

Tabulka **4.2**: Přehled vhodných procesorů s implementovaným řadičem CAN a USB 1/2

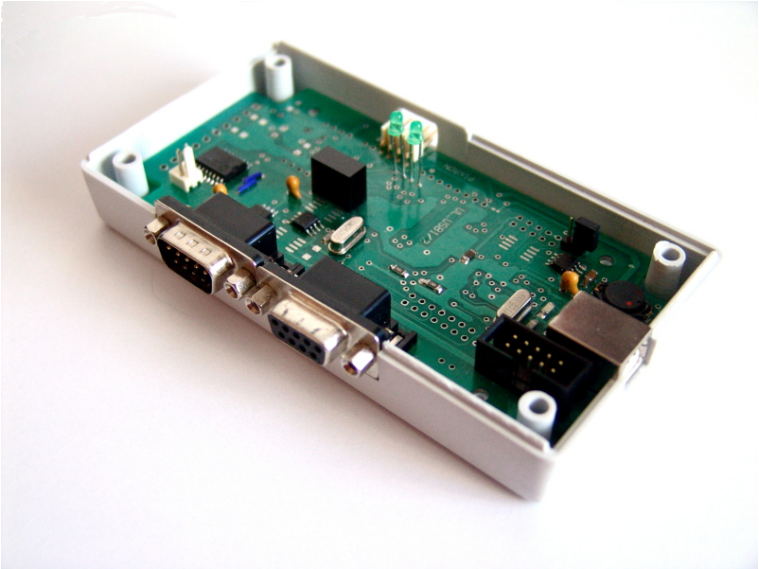
Název	Jádro	Paměť [kB] Flash	RAM	Počet I/O	f_{MAX} [MHz]	Pouzdro	Poznámka	Cena/ks
STM STM32F103VB	Cortex-M3	128	20	80	72	LQFP100	USB2.0 FS	9,59 $/1
STM STM32F103VC	Cortex-M3	256	38	80	72	LQFP100	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103VD	Cortex-M3	384	64	80	72	LQFP100	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103VE	Cortex-M3	512	64	80	72	LQFP100	USB2.0 FS, řadič externí paměti	14,98 $/1
STM STM32F103ZC	Cortex-M3	256	48	112	72	LQFP144	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103ZD	Cortex-M3	384	64	112	72	LQFP144	USB2.0 FS, řadič externí paměti	N/A
STM STM32F103ZE	Cortex-M3	512	64	112	72	LQFP144	USB2.0 FS, řadič externí paměti	16,15 $/1
STM STR710FZ1	ARM7TDMI-S	144	32	50	48	LBGA144, TQFP144	USB2.0 FS	14,50 $/1
STM STR710FZ2	ARM7TDMI-S	272	64	50	48	LBGA144, TQFP144	USB2.0 FS	16,05 $/1
STM STR750FV1	ARM7TDMI-S	144	16	60	72	TQFP100	USB2.0 FS	9,56 $/1
STM STR750FV2	ARM7TDMI-S	272	16	60	72	TQFP100	USB2.0 FS	11,12 $/1
STM STR911FAM42	ARM966E-S	288	96	40	96	LQFP80	USB2.0 FS	10,93 $/1
STM STR911FAM44	ARM966E-S	544	96	40	96	LQFP80	USB2.0 FS	12,20 $/1
STM STR911FAM46	ARM966E-S	1152	96	40	96	LQFP80	USB2.0 FS	9,56 $/563
STM STR911FAM47	ARM966E-S	2176	96	40	96	LQFP80	USB2.0 FS	11,54 $/563
STM STR911FAW42	ARM966E-S	288	96	80	96	LQFP128	USB2.0 FS	11,31 $/1
STM STR911FAW44	ARM966E-S	544	96	80	96	LQFP128	USB2.0 FS	12,84 $/1
STM STR911FAW46	ARM966E-S	1152	96	80	96	LQFP128	USB2.0 FS	10,21 $/416
STM STR911FAW47	ARM966E-S	2176	96	80	96	LQFP128	USB2.0 FS	12,23 $/416
STM STR911FM44	ARM966E-S	544	96	40	96	LQFP80	USB2.0 FS	19,53 $/1
STM STR912FAW42	ARM966E-S	288	96	80	96	LQFP128	USB2.0 FS, Ethernet	12,19 $/1
STM STR912FAW44	ARM966E-S	544	96	80	96	LQFP128	USB2.0 FS, Ethernet	13,32 $/1
STM STR912FAW46	ARM966E-S	1152	96	80	96	LQFP128	USB2.0 FS, Ethernet	11,05 $/416
STM STR912FAW47	ARM966E-S	2176	96	80	96	LQFP128	USB2.0 FS, Ethernet	12,78 $/426
STM STR912FAZ42	ARM966E-S	288	96	80	96	LFBGA144	USB2.0 FS, Ethernet	12,72 $/1
STM STR912FAZ44	ARM966E-S	544	96	80	96	LFBGA144	USB2.0 FS, Ethernet	14,92 $/1
STM STR912FAZ46	ARM966E-S	1152	96	80	96	LFBGA144	USB2.0 FS, Ethernet	11,13 $/869
STM STR912FAZ47	ARM966E-S	2176	96	80	96	LFBGA144	USB2.0 FS, Ethernet	12,80 $/891

Tabulka **4.3**: Přehled vhodných procesorů s implementovaným řadičem CAN a USB 2/2

Název	Jádro	Paměť [kB] Flash	RAM	Počet I/O	f_{MAX} [MHz]	Pouzdro	Poznámka	Cena@ks
Atmel AT91SAM7S128	ARM7TDMI-S	128	32	32	55	QFN64, LQFP64		9,22 $/1
Atmel AT91SAM7S256	ARM7TDMI-S	256	64	32	55	QFN64, LQFP64		11,31 $/1
Atmel AT91SAM7SE256	ARM7TDMI-S	256	32	88	48	LQFP128, LFBGA144	USB2.0 FS, řadič externí paměti SDRAM	15,07 $/1
Atmel AT91SAM7S512	ARM7TDMI-S	512	64	32	55	QFN64, LQFP64		15,16 $/1
Atmel AT91SAM7SE512	ARM7TDMI-S	512	32	88	48	LQFP128, LFBGA144	USB2.0 FS, řadič externí paměti SDRAM	15,27 $/1
Atmel AT91FR40162S	ARM7TDMI-S	2048	256	32	75	TFBGA121		23,40 $/1
NXP LPC2144	ARM7TDMI-S	128	16	45	60	LQFP64	USB2.0 FS	8,95 $/1
NXP LPC2146	ARM7TDMI-S	256	32	45	60	LQFP64	USB2.0 FS	10,68 $/1
NXP LPC2148	ARM7TDMI-S	512	32	45	60	LQFP64	USB2.0 FS	11,88 $/1
NXP LPC2888/D1	ARM7TDMI-S	1024	64	85	60	TFBG180	USB2.0 HS, řadič externí paměti SDRAM/FLASH	15,98 $/1
OKI ML67Q5250	ARM7TDMI-S	128	55	43	32	LFBGA144	USB2.0	N/A
OKI ML69Q6203	ARM946E-S	512	128	87	120	LFBGA272	USB2.0 HS, ATAPI/IDE	34,11 $/1
STM STR711FR1	ARM7TDMI-S	144	32	30	50	LFBGA64, TQFP64	USB2.0 FS	11,40 $/1
STM STR711FR2	ARM7TDMI-S	272	64	30	50	LFBGA64, TQFP64	USB2.0 FS	13,05 $/1
STM STR751FR1	ARM7TDMI-S	144	16	38	60	TQFP64	USB2.0 FS	9,30 $/1
STM STR751FR2	ARM7TDMI-S	272	16	38	60	TQFP64	USB2.0 FS	10,76 $/1

Tabulka **4.4**: Přehled vhodných procesorů s řadičem USB bez řadiče CAN

5 Použitý hardware



Obrázek **5.1**: Dodaný hardware společnosti PiKRON

Pro vývoj a testování byla použita deska UL_USB1 společnosti PiKRON (obrázek **5.1**), která využívá procesor Philips LPC2148. Tento procesor je 32bitový s 16bitovým režimem Thumb[9], používá jádro ARM7TDMI-S a disponuje možnostmi trasování kódu pro ladění. Používá vysokorychlostní FLASH paměť s kapacitou 512 kB a statickou operační paměť s kapacitou 32 kB (+ 8 kB paměti používané pro DMA[10] operace). Mezi jinými periferiemi má zabudované sériové rozhraní, které bylo využíváno při vývoji pro nahrávání firmware[11] a ladění, rozhraní sběrnice USB používané pro komunikaci s ovladačem LinCAN, čítače pro časové zpoždění a watchdog pro zajištění správné funkčnosti. Další informace k procesoru jsou dostupné v [14].

Procesor je taktován frekvencí 12 MHz, kterou vnitřní PLL násobičkou[12] zvyšuje na 48 MHz. USB subsystém pracuje v režimu Full Speed (12 Mbit/s), využívá 2 kB paměti RAM pro zprávy a je schopen navíc používat 8 kB paměti RAM pro DMA operace.

[9] Podrobné informace v [14], kapitola 1.6
[10] Přímý přístup do paměti (Direct Memory Access)
[11] Programové vybavení embedded zařízení
[12] Násobička frekvence s fázovým závěsem (Phase Locked Loop)

V dnešní nabídce existují procesory se zabudovaným rozhraním sběrnice CAN, v době návrhu desky UL_USB1 však ještě nebyly k dispozici. Na desce je proto k procesoru LPC2148 připojen řadič Philips SJA1000T.

Řadič SJA1000T umí pracovat v režimu PeliCAN. Tento režim přináší lepší možnosti diagnostiky chyb, rozlišuje více chybových stavů a má přepisovatelná počítadla chyb. Režim dále přináší možnost jednorázového odesílání zpráv (single-shot transmission), kdy v případě přerušení vysílání je zpráva zahozena, režim odposlechu bez interakce se sběrnicí (listen only mode), automatickou detekci rychlosti sběrnice, příjem vlastních zpráv (self reception) a rozšířené možnosti filtru zpráv.

Schéma desky UL_USB1 je uvedeno v příloze B a je také uloženo v souboru ul_usb1.pdf na přiloženém CD-ROM. Reálné zapojení se od schématu liší v propojení pinu ALE a $\overline{CS}$ na SJA1000T s pinem 25 na portu 0 procesoru LPC2148. Původní propojení pinu ALE na SJA1000T s pinem 23 na portu 0 LPC2148 kolidovalo s druhou funkcí pinu, V_{BUS}, indikující přítomnost napájení na sběrnici USB. Z pohledu ovládání řadiče sběrnice CAN není v tato změna kritická. Z časových diagramů uvedených v [15] vychází korelace mezi signály ALE a $\overline{CS}$ při ovládání jednoho zařízení. K této změně bylo přihlédnuto při programování nízké úrovně komunikace s řadičem (zápis a čtení registrů).

6 Softwarové řešení ovladače

6.1 Psaní ovladačů pro OS[13] Linux

Ovladač, jak už název napovídá, slouží k ovládání periferií a komponent počítače a ke komunikaci mezi ním a uživateli. Množství zařízení, která lze připojit k osobnímu počítači, stále roste a je nutné připravovat ovladače pro umožnění jejich funkce v operačním systému. Linux jako operační systém se samozřejmě neomezuje pouze na osobní počítače, ale může fungovat i na různých embedded zařízeních, ve kterých je však také potřeba přítomnosti specializovaných ovladačů.

Ovladače jsou rozšiřující moduly jádra OS pracující v jeho paměťovém prostoru (kernel space). Mohou využívat služeb a subsystémů poskytovaných jádrem. Pro správnou funkčnost celého systému musí být začleněny do struktur jádra a splňovat pravidla daná jeho konceptem.

Zatímco uživatelské programy jsou při chybě ukončeny operačním systémem, ovladače jsou jeho součástí, jakákoli chyba v nich má vliv na celý systém a může způsobit jeho nestabilitu. Na ovladače jsou kladeny zvláštní požadavky na bezpečnost, velikost používaného paměťového prostoru, efektivitu kódu a možnosti mnohonásobných přístupů.

Ovladače pro Linux se dělí na tři kategorie podle formy rozhraní poskytnutého uživatelskému prostoru (user space):

- Znaková zařízení
  Ke znakovým zařízením je přistupováno jako k toku bytů (znaků) tak, jak to funguje například v souborech. Tato zařízení implementují přinejmenším funkce otevření (`open`), zavření (`close`), čtení (`read`), zápis (`write`).
  Typická znaková zařízení neumožňují oproti souborům vyhledávání v obsahu a slouží tedy jako datové kanály. Mezi jejich zástupce patří textové konzole (*/dev/console*) a sériové linky (*/dev/ttyS0*, atd.). Existují ale také výjimky, kde je možná určitá forma vyhledávání a posunů. Jsou jimi například frame grabbery, zařízení umožňující sejmout právě zobrazovaná data.

- Bloková zařízení
  Jsou podobná znakovým zařízením, ale v mnoha Unixových systémech jsou možné jen přenosy dat po blocích 512 (nebo vyšších mocninách čísla 2) bytů. Linux však umožňuje práci s blokovými zařízeními jako

[13] Operační systém

se znakovými, tj. po znacích, rozdíl mezi nimi je tedy ve vnitřní reprezentaci dat. Typickým zástupcem blokových zařízení jsou diskové paměťové jednotky.

- Síťová zařízení
  Jakákoli síťová komunikace probíhá přes síťová zařízení. Ta mohou být propojena s existujícím hardwarem, nebo mohou být čistě softwarová, jako například *loopback* zařízení. Rozhraní s uživatelským prostorem je zodpovědné za příjem a odesílání datových paketů. Síťové zařízení neví nic o jednotlivých připojeních, pouze operuje s pakety. Zařízení tohoto typu nemají o sobě záznamy ve složce */dev*, jak je tomu u ostatních typů, ale namísto toho se k nim přistupuje speciálními metodami přes unikátní názvy (jako například eth0).

Standardní ovladače zařízení musí být zavedeny do jádra, aby bylo možné využívat jejich funkcí. Jejich zavedení probíhá většinou po startu systému nebo jako reakce na požadavek oprávněného uživatele. Pro periferie používající rozhraní, která umožňují funkci *hotplug*[14] často v systému existují procesy, které dokážou příslušný ovladač zavést automaticky po jejich připojení bez zásahu uživatele.

K vytvoření zařízení výše uvedených typů dojde až po načtení a zavedení ovladačů. Každý ovladač potom umožní komunikaci mezi uživatelským prostorem a koncovým zařízením.

Speciální skupinou ovladačů jsou virtuální ovladače. Koncové zařízení nemá fyzickou podstatu, ale bývá to nejčastěji struktura uložená operační paměti, se kterou aplikace v uživatelském prostoru interagují.

6.1.1 Komunikace s USB zařízeními

Pro práci ovladačů je možné využít různých subsystémů jádra. Jedním z nich jsou i služby pro komunikaci s rozhraním USB.

Ovladače zařízení jsou programový kód a jako takové potřebují pro svou funkčnost využívat zdroje systému, jedním z nich je i operační paměť. Z povahy USB zařízení vychází, že nemusí být vůbec připojeny při spuštění počítače a ani za celou dobu jeho běhu, proto by bylo zbytečné příslušné ovladače zavádět bez jejich přítomnosti, aby pouze konzumovaly paměť počítače bez zjevného užitku.

Naproti tomu, pokud dojde k připojení zařízení, je třeba daný ovladač zavést automaticky bez zásahu uživatele (který nemá většinou ani oprávnění tento úkon provést). Z těchto důvodů existuje mechanismus registrace

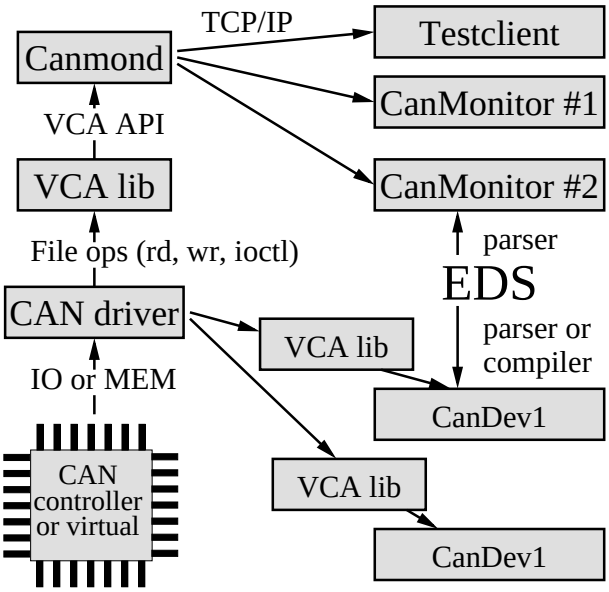
[14] Připojení a odpojení za běhu hostitelského počítače

identifikátorů jednotlivých zařízení, která ovladač podporuje. Výsledkem je publikování této informace do systému, který pak i bez zavedeného ovladače dokáže rozhodnout, který ovladač přísluší danému zařízení, a zavést jej.

Ovladače komunikují s USB zařízeními prostřednictvím objektů, nazývaných URBy (USB Request Block). Po alokaci URBu v paměti s ním můžeme pracovat. Pro úspěšné vyřízení je třeba nastavit zařízení, které bude požadavek vyřizovat, směr komunikace, endpoint, se kterým chceme komunikovat, a jeho typ a přiřadit k němu znakové (bytové) pole s daty, která chceme přenášet, případně prázdné pole, které se naplní přijatými daty. Následně můžeme URB odeslat a po jeho vyřízení nebo chybě dostaneme vyrozumění o stavu prostřednictvím jeho stavových proměnných.

6.2 Ovladač LinCAN

LinCAN je ovladač zařízení komunikujících se sběrnicí CAN pro operační systém Linux a jeho modifikace pro funkci v reálném čase (RT-Linux) od verze jádra 2.2. Je součástí širší programové základny CAN/CANopen vyvíjené jako součást projektu OCERA. Ovladač obsahuje podporu pro více než deset komunikačních karet s různými čipy a různými způsoby přístupu, přímo podporuje standardní integrované obvody – Intel i82527, Phi-



Obrázek **6.1**: Použití ovladače LinCAN[13]

lips 82c200 a Philips SJA1000 (ve standardním režimu i v režimu PELICAN) – používané pro komunikaci na sběrnici CAN, které dokáže ovládat na úrovni jejich registrů, a také nabízí virtuální zařízení pro testovací účely. Mezi podporované karty patří např. P-CAN série vyráběná firmou UNICONTROLS. Ovladač umožňuje v podobě znakových zařízení přístup k frontám zpráv pro jednotlivá rozhraní sběrnice CAN. V uživatelském prostoru tato zařízení mohou využívat další komponenty, jako například CANOPEN a VCALIB, které mj. poskytují rozhraní pro komunikaci s dalšími aplikacemi formou klient/server a poskytují GUI[15] pro výměnu CANových zpráv. (Obrázek **6.1**)

Ovladač implementuje mimo standardních metod také funkce *select*, *poll* a *fasync* a nabízí blokující (O_SYNC) i neblokující (O_NONBLOCK) přístup. Popis těchto metod lze nalézt v [1], kapitole 3.

6.2.1 Komunikace s uživatelským prostorem

Komunikační objekty vytvořené ovladačem LINCAN jsou z pohledu operačního systému znaková zařízení, komunikace s nimi probíhá prostřednictvím standardních požadavků *read/write* přenášením bloků s velikostí a obsahem struktury `struct canmsg_t`, které mají význam jednotlivých CANových zpráv. Komunikační objekty jsou v systému dostupné pod standardními názvy */dev/can0*, *dev/can1, atd.* a podporují všechny standardní souborové operace *open*, *close*, *read*, *write*, *select* a *ioctl*. Popis operací lze nalézt v [1], kapitole 3 a v [13], kapitole 1.3.2.

```
struct canmsg_t {
  int flags;
  int cob;
  unsigned long id;
  canmsg_tstamp_t timestamp;
  unsigned short length;
  unsigned char data[CAN_MSG_LENGTH];
} PACKED;
```

- `int flags`
  Toto pole obsahuje příznaky přenášené zprávy. MSG_RTR způsobí vyslání žádosti o rámec (kapitola 2.3.2), MSG_EXT způsobí použití rozšířeného identifikátoru (kapitola 2.3) a MSG_OVR slouží k rychlému

[15] Grafické rozhraní (Graphical User Interface)

naznačení zaplnění příslušné fronty zpráv. Odesílané zprávy mohou být po úspěšném odeslání na sběrnici zaslány zpět lokálním klientům prostřednictvím nastaveného příznaku MSG_LOCAL.

- `int cob`
  Značí číslo komunikačního objektu. V budoucnu může sloužit k serializaci přijatých zpráv do front.

- `unsigned long id`
  Identifikátor přenášené zprávy.

- `canmsg_tstamp_t timestamp`
  Slouží pro ukládání času příjmu zprávy.

- `unsigned short length`
  Vyjadřuje délku přenášených dat, může obsahovat hodnoty 0 až 8.

- `unsigned char data[CAN_MSG_LENGTH]`
  Bytové pole obsahující data zprávy.

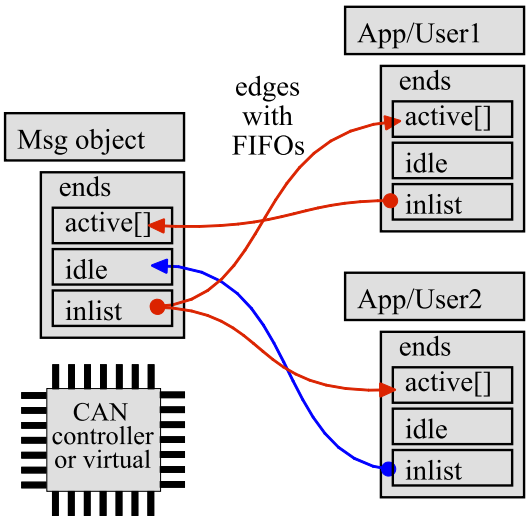
6.2.2 Implementace front zpráv

V ovladači je vyřešeno předávání zpráv mezi uživatelským rozhraním a zařízením prostřednictvím front. Fronty zajišťují plynulost komunikace se sběrnicí CAN. Protože může existovat mnoho uživatelů připojených k jednomu komunikačnímu objektu, existuje i více front a to v obou směrech. Celý systém předávání zpráv tak lze vyjádřit orientovanými hranami mezi uživateli a komunikačními objekty jednotlivých zařízení. Takto je navržena vrstva nad jednotlivými frontami. Fungování hran je naznačeno v obrázku **6.2**.

Práce s frontami je realizována tak, aby nebylo potřeba používat synchronizaci přístupu k jednotlivým slotům. Toho je dosaženo atomickými operacemi pro práci s příznakovými bity *set_bit*, *clear_bit*, *test_bit*, *test_and_set_bit* a *test_and_clear_bit*.

Po vložení zprávy uživatelem je zpráva umístěna do příslušné fronty. Pokud není na straně zařízení právě zpracovávána žádná jiná odchozí zpráva, provede se příslušná funkce, která zprávu převezme a zpracuje. V případě, že již probíhá zpracování některé předchozí zprávy, zůstane nová zpráva ve frontě a dále je již požadavek na její zpracování starostí kódu pro danou periferii. Většinou se kontrola na další zprávy čekající ve frontě provádí v přerušení při dokončení odesílání.

Přijetí zprávy je nejčastěji indikováno přerušením od zařízení, v jehož obsluze je zpráva předána do všech uživatelských front.



Obrázek **6.2**: Fungování hran v ovladači LinCAN[13]

Pro fronty odchozích zpráv jsou na straně zařízení implementovány funkce:

- ```
  int canque_test_outslot(struct canque_ends_t *
      qends, struct canque_edge_t **qedgep, struct
      canque_slot_t **slotp);
  ```

  Funkce vezme hranu s nejvyšší prioritou a z ní přijme nejstarší zprávu. Návratová hodnota menší než 0 znamená, že není k dispozici žádná zpráva, hodnoty větší nebo rovny 0 značí úspěšné převzetí zprávy.

- ```
  int canque_free_outslot(struct canque_ends_t *
      qends, struct canque_edge_t *qedge, struct
      canque_slot_t *slot);
  ```

  Funkce uvolní slot zprávy, který byl v předchozím kódu získán příkazem `canque_test_outslot()`. Bývá volána po odeslání zprávy. Návratová hodnota značí, zda vstupní strana byla informována o změně stavu slotu.

- ```
  int canque_again_outslot(struct canque_ends_t *
      qends, struct canque_edge_t *qedge, struct
      canque_slot_t *slot);
  ```
  ```

Funkce vrátí slot se zprávou do čekající fronty. Může být zavolána při dočasných problémech s odesíláním zprávy. Funkce vždy skončí úspěšně

Při přijetí zprávy je pro předání uživatelským hranám implementována funkce:

- `int canque_filter_msg2edges(struct canque_ends_t *qends, struct canmsg_t *msg);`

  Funkce předá zprávu všem odchozím hranám, které přijímají zprávy s daným identifikátorem. Návratová hodnota značí počet hran, na které byla zpráva odeslána.

Další informace k implementaci front v ovladači LinCAN jsou k dispozici v dokumentaci [13] a ve zdrojovém kódu ovladače.

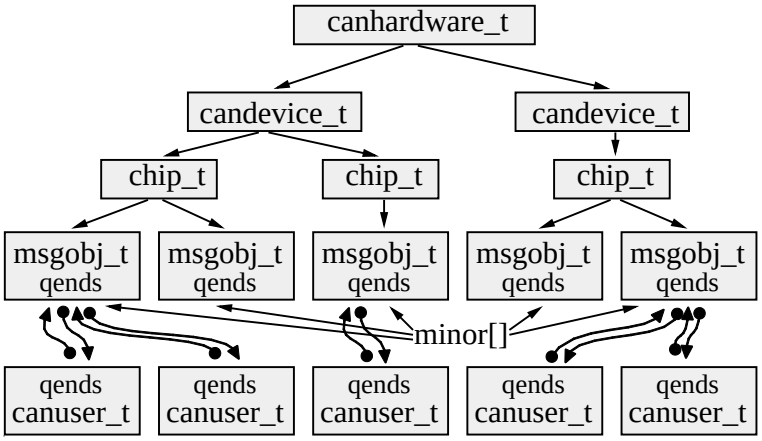
6.2.3 Hierarchie zařízení v ovladači

Jednotlivá zařízení mají v ovladači přiřazené datové struktury. Tyto struktury popisují hierarchii zařízení, která mohou mít integrovaných i více různých komunikačních čipů, každý s několika komunikačními objekty. Tato hierarchie je naznačena v obrázku **6.3**. Jednotlivým komunikačním objektům jsou přiřazena znaková zařízení a obdrží svoje *minor*[16] číslo. Každý uživatel může být připojen pouze k jednomu komunikačnímu objektu, ale každý komunikační objekt může mít více připojených uživatelů.

Struktury candevice_t a chip_t obsahují seznamy funkcí hwspecops a chipspecops. Tyto seznamy obsahují odkazy na běžné funkce zařízení, jako je spuštění zařízení a jeho zastavení, alokace jeho zdrojů a jejich uvolnění, spuštění běhu čipu, zastavení, apod. Mimo to samozřejmě obsahují funkce pro komunikaci.

Funkce ve struktuře hwspecops jsou většinou pro každé zařízení jiné, udávají totiž i mj. počet a typ čipů na desce ovládaného zařízení. Naproti tomu funkce jednotlivých čipů jsou většinou stejné, protože většina ovladačem podporovaných zařízení pracuje na základě čipů, které jsou výše uvedeny mezi standardními. Jejich funkce byly proto přesunuty do oddělených souborů.

[16] číselný identifikátor zařízení v operačním systému



Obrázek **6.3**: Hierarchie zařízení v ovladači LinCAN

6.3 Přidání podpory zařízení USBCAN do ovladače LinCAN

Ovladač pro zařízení, které je popsáno v kapitole 5, se v mnohém vymyká standardním zařízením podporovaným ovladačem LinCAN. Jednou z podstatných odlišností je samozřejmě použití sběrnice USB, neboť všechna podporovaná zařízení až dosud používala nejvýše sběrnici PCI. U té nehrozí připojování a odpojování zařízení za běhu, proto ani tyto problémy dosud nemusely být řešeny. Původní ovladač v několika případech neměl implementovány rutiny pro odstraňování datových struktur z paměti, protože k tomu docházelo jedině při jeho ukončování, kdy se uvolňovala alokovaná paměť jako celek.

Pro nově řešené ovládání komunikace s řadičem bylo k dispozici několik možností. Jednou z nich bylo využití rutin pro standardní čip SJA1000T a řízení hardwaru zasíláním a čtením obsahu jeho registrů přímo přes USB. Tato možnost byla lákavá a první kroky vedly k její realizaci.

Přestože komunikace fungovala, přístupy po osmibitových blocích jsou vhledem ke koncepci komunikačního protokolu sběrnice USB velmi nehospodárné a čekání na odpovědi při čtení každého registru znamená neúnosná zpoždění.

USB sběrnice je především optimalizovaná na plynulý přenos proudu dat nebo paketů, například na disk, do reproduktorů nebo z kamery. Při uvažování přenosu zpráv výše zmíněnou metodou tak dochází k zatěžování sběrnice velmi častými přenosy malých objemů dat a efektivita převodníku i sběrnice prudce klesá. Popsaný způsob byl proto nakonec zavrhnut.

K finální implementaci byla vybrána metoda komunikace se zařízením

prostřednictvím datových bloků obsahujících jednotlivé CANové zprávy a řízení převodníku pomocí vendor požadavků[17]. Kód ovladače USBCAN tak nepoužívá funkce pro standardní komunikační čip SJA1000 i přesto, že jej převodník obsahuje.

6.3.1 Zavedení ovladače pro USB zařízení

Ovladače USB zařízení obsahují oproti standardním ovladačům speciální funkce podporující připojování a odpojování za běhu. V popisovaném ovladači to jsou následující funkce:

```
static int usbcan_probe(struct usb_interface *
    interface, const struct usb_device_id *id);
static void usbcan_disconnect(struct usb_interface *
    interface);
```

Odkazy na tyto funkce jsou dále vloženy do struktury typu `struct usb_driver` spolu s názvem ovladače a tabulkou podporovaných zařízení.

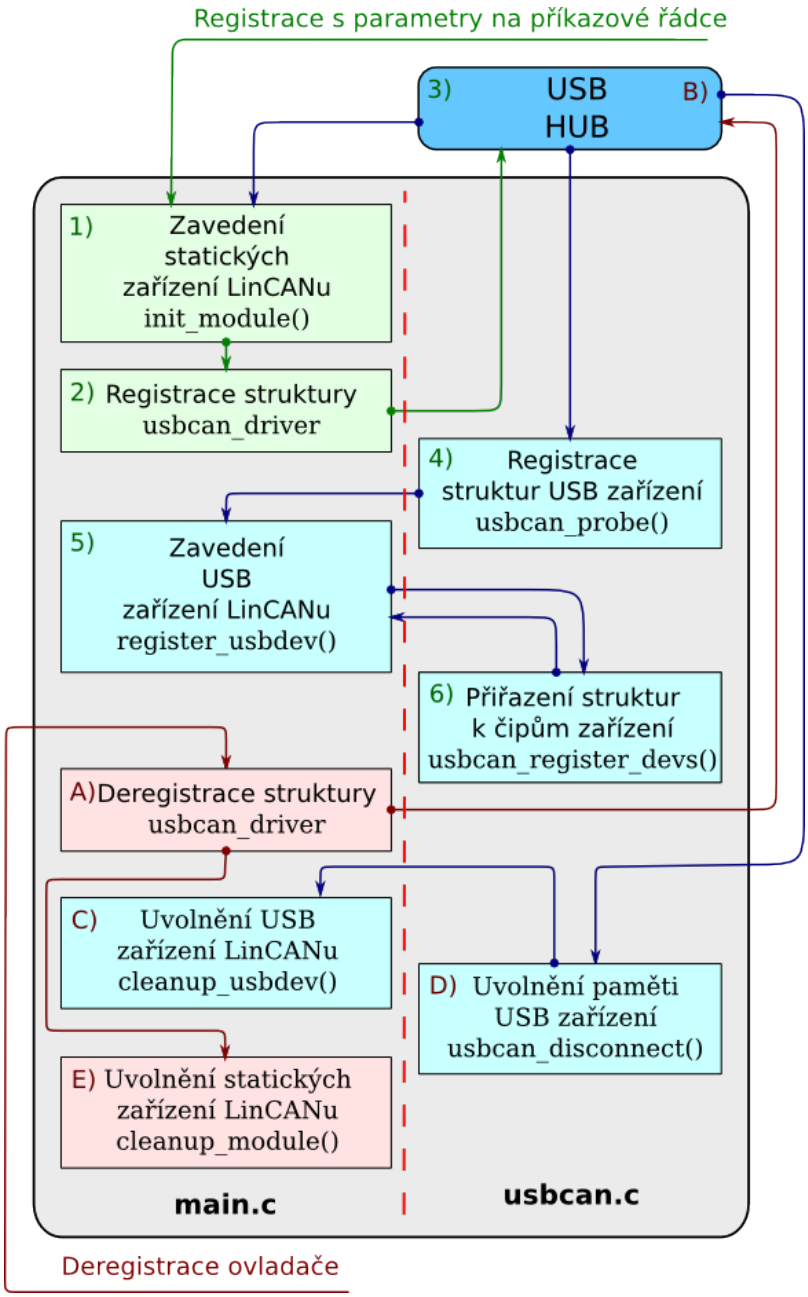
```
static struct usb_driver usbcan_driver = {
  .name = "usbcan",
  .id_table = usbcan_table,
  .probe = usbcan_probe,
  .disconnect = usbcan_disconnect,
};
```

Položka `usbcan_table` obsahuje parametry identifikující jednotlivá podporovaná zařízení. V tomto případě jde o dvojici čísel výrobce (Vendor ID) a zařízení (Product ID). Pro testovací účely byla čísla vymyšlena, pro produkci výrobku je však třeba mít registrované číslo výrobce u organizace USB IF.

Struktura `usbcan_driver` musí být v posledním kroku registrována do usb subsystému při zavedení ovladače do paměti příkazem `usb_register()`. Při ukončování běhu ovladače musí být zavolána funkce `usb_deregister()`.

Funkce `usb_probe()` je volána po připojení zařízení ke sběrnici USB, poté co proběhla jeho enumerace. Funkce by měla ověřit, že zařízení, které je uvedeno v parametrech, je skutečně ovladačem podporované. Pokud registrace proběhne vpořádku, funkce vrátí 0, jinak musí vrátit chybový kód. Uvnitř funkce by měly proběhnout pouze nejnutnější operace, protože probíhá v kontextu vlákna USB rozbočovače, který vyřizuje požadavky i dalších

[17] Konfigurační požadavky přenášené prostřednictvím EP0



Obrázek **6.4**: Zavedení a ukončení ovladače

zařízení.

Postup zavedení a ukončení ovladače je zřejmý z obr. **6.4**.

Zavedení ovladače:

1. Při zavádění ovladače LinCAN jsou všechny parametry statických zařízení předávány pomocí parametrů příkazové řádky. Tuto metodu nelze celou využít pro registraci USB zařízení, proto musely být pozměněny některé části kódu. Změny, aby neovlivňovaly původní kód, byly shrnuty do níže popsané nové funkce `register_usbdev()` v souboru *main.c*.

2. Po provedení registrace zařízení zadaných v příkazové řádce je (v případě kompilace ovladače s podporou zařízení USBCAN) zavolána funkce `int usbcan_init()` provádějící registraci struktury `usbcan_driver`.

3. Po registraci struktury `usbcan_driver` zavolá při přítomnosti zařízení v systému USB subsystém funkci `usbcan_probe()`.

4. Funkce `usbcan_probe()` provede registraci příslušných datových struktur v závislosti na poskytnutých informacích o zařízení a zavolá rutinu `register_usbdev()`.

5. Ve funkci `register_usbdev()` dojde k registraci zařízení v hierarchii ovladače LinCAN a vytvoření příslušného souboru ve složce */dev*.

6. Pokud připojené zařízení podporuje více komunikačních objektů, poskytne vyšší (sudý) počet bulk endpointů, se kterými lze komunikovat. Ovladač na tuto možnost zareaguje vytvořením většího počtu čipů přítomných v zařízení a pracujících nezávisle. Ke každému čipu potom přísluší vlastní datová struktura typu `struct usbcan_usb`. Pro správnou funkčnost ovladače převodníku je nutné do každého čipu vložit ukazatel na tuto datovou strukturu. K tomu slouží rutina předávaná jako parametr funkce `register_usbdev()`, která se jmenuje `usbcan_register_devs()`.

Pokud již byl ovladač zaveden v době připojení periferie, začíná proces registrace od bodu 3.

Při odpojení zařízení od sběrnice USB dochází k procesu deregistrace příslušného zařízení tak, jak je naznačeno v obrázku **6.4**, v bodech B až D. USB subsystém zavolá funkci `usbcan_disconnect()`, která na svém začátku odstraní příslušné znakové zařízení, aby se zamezilo další komunikaci s uživatelským prostorem a poté uvolní všechny alokované prostředky. Při odpojení USB zařízení nedojde k ukončení běhu celého ovladače.

Pokud je běh operačního systému ukončován nebo přišel požadavek na ukončení běhu ovladače, proběhne proces popsaný v bodech A až E. Deregistrací struktury USB zařízení (A) dojde k výše popsanému ukončení běhu USB zařízení (B–D), které je následováno uvolněním statických zařízení a alokované paměti (E).

```
struct candevice_t* register_usbdev(const char *
    hwname,void *devdata,void (*chipdataregfnc)(
    struct canchip_t *ch,void *data))
```

- `const char *hwname`
  Text identifikující typ zařízení a jeho registrační funkci ve stuktuře dostupných zařízení v souboru *boardlist.c*. Metoda registrace přes textový název byla zvolena pro shodu s původním způsobem registrace.

- `void *devdata`
  Ukazatel na strukturu obsahující parametry USB zařízení a další potřebné proměnné pro běh ovladače.

- `void (*chipdataregfnc)(struct canchip_t *ch,void * data)`
  Ukazatel na funkci, která zaregistruje příslušné USB zařízení k dané struktuře čipu.

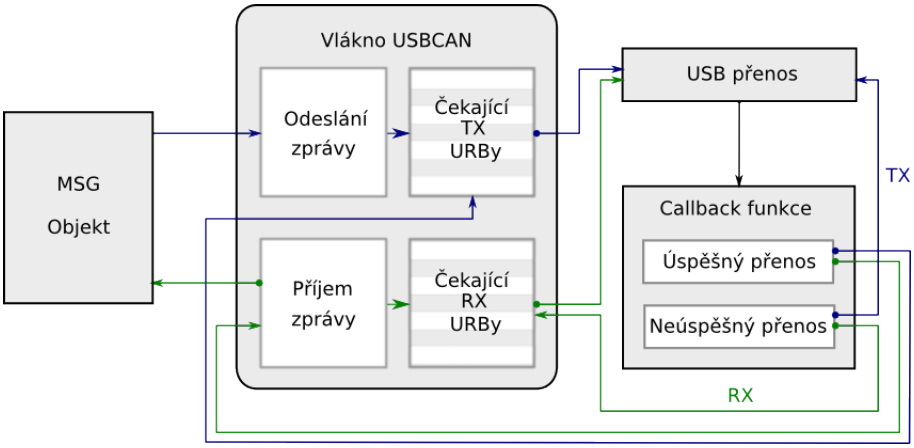
Návratovou hodnotou funkce je ukazatel na strukturu nově registrovaného zařízení v případě úspěchu a NULL v případě chyby.

6.3.2 Zpracování toku zpráv

Při komunikaci ovladače s převodníkem jsou používány bulk endpointy pro přenos CANových zpráv a řídící EP (vendor požadavky) pro dodatečnou konfiguraci zařízení.

Jak je popsáno v kapitole 6.1.1, komunikace probíhá prostřednictvím požadavkových bloků (URB). Pro komunikaci v obou směrech, která je znázorněna na obrázku **6.5**, je potřeba vytvořit nejméně dva URBy, každý s jiným definovaným směrem toku dat. Po dokončení URBu je do jeho stavových proměnných uložena informace o úspěchu nebo neúspěchu přenosu. Zároveň je zavolána callback funkce (ukazatel na ni je uložen mezi parametry URBu), která má provést požadované kroky.

Při odesílání nebo příjmu zpráv ovladačem LinCAN je možné, že dojde



Obrázek **6.5**: Přenos zpráv v ovladači

k uspání procesu. *Callback* funkce volaná při vyřízení URBu však probíhá v kontextu přerušení a není tedy dovoleno v ní usnout. Přijaté zprávy proto nemohou být předávány k dalšímu zpracování rutinami LinCANu uvnitř *callback* funkce, která tak pouze nastavuje příznakové bity stavových proměnných ovladače a ty jsou dále zpracovávány mimo kontext přerušení.

Přenášené datové bloky

CANové zprávy jsou v ovladači vnitřně reprezentovány strukturou `struct canmsg_t`, která obsahuje veškerá potřebná pole zprávy, ale také informace, které slouží pouze ovladači LinCAN. Složení struktury a velikosti jednotlivých proměnných jsou platformově závislé, proto struktura jako celek není vhodná pro přímý přenos po sběrnici USB. Jedním ze způsobů zajišťování bezchybného přenosu dat je uložení dat do pevně dané struktury, jakou je bytové pole. Jednotlivé vícebytové proměnné jsou do pole ukládány ve formátu little-endian prostřednictvím maker `cpu_to_le16()` a `cpu_to_le32()` a čteny za pomoci maker `le16_to_cpu()` a `le32_to_cpu()`. Zprávy jsou tedy přenášeny jako datové bloky. Délka datových bloků byla na základě minimální potřebné délky dat a možností volby délky bufferu v připojeném převodníku zvolena 16 B. V bloku této délky lze přenést celou zprávu a zbývá místo pro budoucí potřeby.

Jednou z vlastností procesorů s architekturou ARM je zarovnání vícebytových proměnných. Tyto procesory neumožňují přistupovat k vícebytovým proměnným (např. `int`, který má délku 4 B), pokud jejich adresa v paměti není násobkem délky proměnné. V přenášeném datovém bloku lze toto

	2 B		2 B	
	1 B	1 B	1 B	1 B
0	Rezervováno	Délka dat	Příznakové bity	
4	Identifikátor			
8	Data	Data	Data	Data
12	Data	Data	Data	Data

Obrázek **6.6**: Struktura datového bloku CANové zprávy pro přenos mezi ovladačem a převodníkem

řešit ukládáním jednotlivých bytů, ze kterých se ve firmwaru procesoru rekonstruují vícebytové proměnné, nebo změnou pořadí proměnných v bloku tak, aby byly zarovnané. V ovladači převodníku byla zvolena druhá možnost a výsledná podoba datového bloku je uvedena v obrázku **6.6**.

Použití vlákna pro zpracování zpráv

Jak bylo výše řečeno, ovladač LinCAN podporuje jak blokující, tak také neblokující přístup k zařízením z uživatelského prostoru. Neblokující přístup znamená potřebu rychlého odesílání zpráv, kdy není žádoucí čekat na jejich vyřízení. Z tohoto důvodu nelze implementovat smyčky čekající na vyřízení zpráv v hlavním vlákně ovladače a je nutné vytvořit nové oddělené vlákno.

Ke spuštění vlákna dochází ve funkci `usbcan_attach_to_chip()`. Tato funkce je volána při prvním požadavku na otevření (metoda *open*) příslušného znakového zařízení. Tím se zamezí zatěžování prostředků počítače, dokud nikdo dané zařízení nepoužívá. Požadavek k ukončení vlákna je vydán ve funkci `usbcan_release_chip()`, která je volána při deregistraci struktur čipu. V závěru před uvolňováním alokovaných struktur je dále vložena smyčka čekající na ukončení vlákna.

Vlákno při svém běhu vykonává funkci `usbcan_kthread()`. V této funkci je třeba zřídit nekonečnou smyčku, která bude vyřizovat veškerou komunikaci. Ve chvílích nečinnosti, kdy neprobíhá přenos zpráv, je vhodné vlákno uspat. Z tohoto důvodu v callback funkci URBů dochází kromě nastavení stavových bitů také ke vzbuzení vlákna, aby mohly být provedeny potřebné operace.

Použití více URBů pro komunikaci

Ovladač převodníku není jediným procesem běžícím v operačním systému. Z tohoto důvodu často dochází k preempci a zdržení přenosů. V době, kdy jsou zpracovávány výsledky dokončeného URBu, je možné že budou při-

cházet další požadavky na přenos dat, které musí být v co nejkratší době vyřízeny. Proto je výhodné umožnit vkládání USB požadavků *do zásoby* a mít tedy více URBů čekajících na vyřízení. Výsledkem je zvýšení plynulosti toku dat při nepatrném zvýšení nároků na USB subsystém a operační paměť.

Požadavky na přenos (URBy), je možné vytvářet vždy při potřebě přenosu, nese to však s sebou určitou režii na alokaci paměti a vytvoření struktur. Proto jsou na úvodu vlákna předalokovány URBy pro příjem i odesílání zpráv, které jsou v případě příchozích URBů pouze znovu odesílány a v případě odchozích URBů jsou jim před odesláním měněny datové bloky obsahující přenášené zprávy. Po alokaci dochází k odeslání všech příchozích URBů, které tak začnou čekat na příjem zprávy. Počet čekajících požadavků byl stanoven na 8 pro odchozí a 8 pro příchozí směr, tyto počty lze změnit prostřednictvím maker `USBCAN_TOT_TX_URBS` a `USBCAN_TOT_RX_URBS` definovaných v hlavičce *usbcan.h*.

Průběh odesílání a příjmu zpráv (obr. 6.5)

Při pokusu o odeslání zprávy přes sběrnici CAN je ovladačem zavolána funkce `usbcan_wakeup_tx()`, uvnitř které dojde k nastavení potřebných stavových bitů pro další vnitřní mechanismy ovladače a ke kontrole, zda je k dispozici nějaký volný odchozí URB. Při úspěchu je vzbuzeno vlákno, které získá slot zprávy a zpracuje data a při neúspěchu zůstává zpráva ve frontě v čekajícím stavu. Po úspěšném vyřízení odchozího URBu je uvolněn slot zprávy, nastaven příznak volné zprávy a provedena kontrola, zda existuje nějaká zpráva čekající ve frontě.

Při příjmu zprávy proběhne ve vlákně převedení obsahu zprávy na strukturu `struct canmsg_t`, která je následně předána připojeným uživatelům a odbavený URB je vrácen do čekajícího stavu.

Neúspěšné přenosy jsou řešeny opakovaným odesláním příchozích i odchozích URBů. Zvláštní situaci tvoří přijetí chybového stavu znamenajícího ukončení URBu. K tomu může dojít pouze během deregistrace ovladače, proto není opakován přenos a URBy jsou ukončeny.

7 Firmware převodníku USBCAN

Pro napsání firmware převodníku bylo využito základní podpory pro mikroprocesory rodiny LPC2xxx, která byla implementována v rámci dalších projektů na katedře řídící techniky a v projektu uLan[18].

Firmware převodníku je psán jazykem C. V programovém kódu je možné díky adaptaci pro architekturu ARM používat také standardní systémové knihovny pro práci s textem, pamětí, apod.

Po zkompilování vznikne binární soubor, který je za pomoci programu *lpc21isp* uploadován přes sériové rozhraní do procesoru.

7.1 Komunikace s řadičem SJA1000T

Pro připojení řadiče SJA1000T k procesoru byla použita standardní vstupně-výstupní brána, proto musela být komunikace řešena na nejnižší úrovni přímo bitovými operacemi. Při programování zápisu a čtení byla vzata v potaz změna v zapojení modulu oproti schématu popsaná v kapitole 5. Při ignorování této změny do programu nefungovala komunikace s řadičem a navíc vznikaly problémy s USB rozhraním.

Řadič je prostřednictvím pinu 11 (MODE) nastaven do režimu INTEL, který má oproti režimu MOTOROLA jiný význam některých pinů, změněny hodnoty některých registrů po resetu a jiné časování úrovní pinů při komunikaci.

Protože nebyla pro komunikaci s řadičem využita žádná standardní sběrnice, muselo být ve firmwaru převodníku vyřešeno i její časování. Časování jednotlivých signálů pro zápis a čtení registrů řadiče je uvedeno v [15], kapitole 10. Uvedené doby trvání jednotlivých fází je možné přímo použít z dokumentace, protože je řadič taktován na stejné frekvenci, jako referenční zapojení, tedy 24MHz. Procesor LPC2148 je taktován na frekvenci 48 MHz, je tedy schopen dosáhnout rychlejších změn hodnot na příslušných pinech. Přestože uvedené minimální doby potřebné pro jednotlivé operace většinou korespondují s dobami změn hodnot na pinech procesoru, byly při experimentech pozorovány chyby v komunikaci a důsledkem toho byly doby pro komunikaci prodlouženy makry *SJA1000_DELAY* a *SJA1000_INIT_DELAY* definovanými v souboru *can.h*. Funkce `can_read()` a `can_write()` pro čtení a zápis registrů jsou umístěné v souboru *can.c*. Soubor dále obsahuje rutinu `can_init()` pro zahájení komunikace s řadičem – počáteční nastavení směru pinů procesoru a reset řadiče.

[18] http://sourceforge.net/projects/ulan/

7.2 Vyšší úroveň komunikace, posílání a příjem zpráv

Pro implementaci ovládání řadiče na vyšší úrovni zahrnující nastavení režimu řadiče (PeliCAN), nastavení registrů pro správnou funkčnost obvodu, nastavení přenosové rychlosti sběrnice a samotný přenos zpráv byly využity součásti ovladače LinCAN, zejména soubory *sja1000p.h* a *sja1000p.c*.

V ovladači LinCAN je práce s řadičem SJA1000 prováděna prostřednictvím přerušení vyvolaných změnami stavů řadiče. Tento způsob práce je umožněn i v modulu UL_USB1, protože je z řadiče k procesoru veden pin, který přerušení indikuje. K obsluze přerušení řadiče však nedochází ihned vyvoláním přerušení procesoru, ale monitorováním daného pinu v hlavní programové smyčce a případným zavoláním obslužné funkce. Tento postup při poměru rychlosti sběrnice CAN (max. 1 Mbit/s) a frekvence procesoru nemá žádný vliv na rychlost komunikace a odpadají tím problémy se synchronizací datových přenosů.

Z ovladače LinCAN byly převzaty nejen funkce pro komunikaci, ale také byl implementován systém front. Systém, který je důkladně popsán v dokumentaci ovladače [13], se může zdát pro účel programovaného zařízení příliš složitý, poskytuje však možnosti plynulého toku zpráv díky jejich frontám a možnosti snadného rozšíření o další řadiče sběrnice CAN v budoucnu. Použité rutiny LinCANu jsou až na drobnosti kompatibilní se použitým ovladačem LinCAN (verze 0.3.3). Jednou z důležitých úprav kódu byla náhrada systému zamykání a synchronizace procesů v ovladači zakazováním a povolováním veškerých přerušení v procesoru. Další změny se týkají změn příkazů používaných pro alokaci paměťového prostoru. Funkce v ovladači používají příkaz `kmalloc()` existující pouze v jádře OS, ve firmware musela být použita standardní verze `malloc()`.

7.3 Komunikace s hostitelem

Standardní komunikace probíhá prostřednictvím USB rozhraní, které používá pro svou funkčnost knihovny usbbase a lpcusb. Pro enumeraci (prvotní identifikaci) zařízení slouží soubor *usb_defs.h* obsahující identifikaci zařízení, popis konfigurace, rozhraní, seznam dostupných endpointů a příslušné textové popisky. Tyto proměnné používá funkce `usb_control_response()` z knihovny usbbase, která obsluhuje systémové požadavky i uživatelské požadavky popsané níže.

Přenos zpráv probíhá v obou směrech prostřednictvím standardního (bulk) endpointu. Procesor podporuje 16 EP, každý v obou směrech, celkem je tedy k dispozici 32 identifikátorů datových kanálů, z nichž 2 slouží pro EP0 (kapitola 3.3.3).

Ovladač převodníku ve svém kódu umožňuje použití více řadičů nebo více komunikačních objektů jednoho řadiče prostřednictvím použití dalších endpointů v převodníku. Počet komunikačních kanálů použitých v jednom zařízení tedy může být až 15. V použitém modulu je k procesoru připojen jeden řadič sběrnice CAN, kterému odpovídá jedno zařízení a jeden EP pro komunikaci.

Funkce `usb_check_events()` zajišťuje detekci přijetí nebo úspěšného odeslání zpráv a následné nastavené příznakových bitů podle endpointu, na kterém nastala změna. V programové smyčce tedy stačí monitorovat příslušné bity pro zjištění stavu přenosu.

Pokud byla z USB rozhraní přijata zpráva, testuje se, zda existuje volný slot ve vnitřní frontě zpráv, což lze zjistit pokusem o jeho získání. V případě existence volného slotu je přijat datový blok USB zprávy, který je popsán v kapitole 6.3.2, data jsou přesunuta do struktury `struct canmsg_t` a slot je následně odeslán do fronty řadiče CAN.

Po spuštění převodníku a vždy při dokončení odesílání je nastaven příznak volného odchozího endpointu. Při jeho detekci je testováno, zda ve frontě zpráv čeká zpráva na odeslání rozhraním USB. Pokud existuje čekající zpráva, je uložena do datového bloku vhodného pro přenos po sběrnici USB a odeslána.

Probíhající komunikace s převodníkem je indikována LED diodami připojenými k univerzálním I/O portům mikroprocesoru.

7.4 Uživatelské požadavky (vendor specific requests)

Kromě toku CANových zpráv je potřeba vyřizovat požadavky na změny konfigurace, které přicházejí prostřednictvím EP0. Díky speciální povaze tohoto EP nelze příjem požadavků realizovat stejně jako u standardních typů EP, jsou ale obslouženy v rutině `usb_control_response()`, která zpracovává výhradně přenosy přes EP0.

Uživatelské požadavky jsou jedním z typů řídících zpráv, které jsou popsány v [7] a [1], kapitole 13.

Zprávy přenášené pomocí řídících endpointů obsahují kromě pole rozlišujícího typ požadavku `bRequestType` také pole `bRequest`, `wValue` a `wIndex`, které umožňují blíže specifikovat požadavek a přenést konfigurační parametry. Pokud funkce `usb_control_response()` určí typ zprávy jako uživa-

telský požadavek, je zavolána obslužná funkce. Ukazatel na tuto funkci je uložen do struktury typu `usb_device` při inicializaci mikroprocesoru.

Požadavky přenášené prostřednictvím řídícího EP mohou obsahovat stejně jako standardní zprávy blok dat. V případě jednoduchých požadavků se obvykle datové části nepoužívají a jejich parametry se vkládají do výše uvedených polí požadavku. Pokud je třeba přenést vyrozumění o úspěšnosti požadavku, je vhodné přenášet požadavek jako požadavek čtení z periferie, kde pole v hlavičce zprávy určují typ požadavku a parametry a datový blok obsahuje vyrozumění o výsledku.

Firmware řadiče implementuje vendor požadavky definované následujícími makry:

- `USBCAN_VENDOR_EXT_MASK_SET`
  provede nastavení rozšířené masky pro příjem zpráv, používá datový blok k přenosu masky, proto je zapotřebí dalšího požadavku k vyrozumění hostitele o vyřízení,
- `USBCAN_VENDOR_EXT_MASK_STATUS`
  zjišťuje výsledek předchozího nastavení masky příchozích zpráv,
- `USBCAN_VENDOR_BAUD_RATE_SET`
  provede nastavení rychlosti sběrnice CAN, používá datový blok k přenosu konfiguračních parametrů sběrnice, proto je zapotřebí dalšího požadavku k vyrozumění hostitele o vyřízení,
- `USBCAN_VENDOR_BAUD_RATE_STATUS`
  zjišťuje výsledek předchozího nastavení rychlosti sběrnice,
- `USBCAN_VENDOR_SET_BTREGS`
  provede nastavení registrů řadiče přímo nastavujících parametry sběrnice CAN,
- `USBCAN_VENDOR_CHECK_TX_STAT`
  kontroluje, zda existují volné sloty ve frontě řadiče,
- `USBCAN_VENDOR_START_CHIP`
  spustí operace řadiče sběrnice CAN,
- `USBCAN_VENDOR_STOP_CHIP`
  zastaví operace řadiče sběrnice CAN.

8 Závěr

Úkolem práce bylo po úvodním rozboru problematiky naprogramovat firmware převodníku mezi sběrnicemi USB a CAN a sestavit přehled vhodných procesorů pro tuto aplikaci. Součástí práce bylo také adaptovat ovladač pro operační systém Linux, který bude s tímto převodníkem spolupracovat.

Modul UL_USB1 vyrobený společností PiKRON a dodaný k testování obsahuje řadič sběrnice CAN v odděleném integrovaném obvodu. Dodaný modul, který používá mikroprocesor Philips LPC2148 a řadič sběrnice CAN Philips SJA1000T, byl navržen v době, kdy byla dostupnost integrovaných řešení mikroprocesoru s řadičem sběrnice CAN omezená. V dnešních mikroprocesorových obvodech postavených na architektuře ARM již jsou, jak se ukázalo při průzkumu trhu těchto obvodů, v převážné většině řadiče sběrnice CAN integrované ve společném pouzdru.

Použití řadiče řady SJA1000 v modulu převodníku inspirovalo k implementaci částí programového kódu ovladače LinCAN, který pro tento obvod obsahuje podporu, do firmware mikroprocesoru. Po prostudování kódu ovladače bylo navíc rozhodnuto o využití dalších částí, především systému front pro přenos jednotlivých zpráv.

Protože komunikace po sběrnici CAN je vpodstatě síťová komunikace, nabízí se možnost implementovat ovladač převodníku jako ovladač síťového zařízení. Ovladače sběrnice CAN, které takto fungují, již existují, jejich nevýhodou je však zatím poměrně vysoká režie pro přenesení jednoho datového paketu vzhledem k velikosti CANové zprávy. Architektura síťových zařízení je totiž optimalizována pro přenos velkých objemů dat, které jsou využívány v mnoha serverových aplikacích. Z tohoto důvodu byl pro implementaci použit ovladač LinCAN, který efektivně zpracovává zprávy a jejich fronty bez zbytečných nároků na zdroje počítače. Ovladač verze 0.3.3 byl rozšířen o podporu nové desky s názvem USBCAN. Zavedení podpory převodníku s sebou přineslo úpravy v některých částech kódu ovladače, který do té doby nepodporoval zařízení připojená přes sběrnici USB.

Při programování převodníku a jeho ovladače bylo nutné vyřešit formu komunikace. Zprávy řídící běh převodníku probíhají přes řídící datový kanál (Control Endpoint), přenos zpráv sběrnice CAN se děje prostřednictvím datových bloků přenášených standardními datovými kanály (Bulk Endpoints) sběrnice USB. Vzhledem k možnosti použití ovladače na počítačích různých architektur s různými délkami datových typů a různým endianingem[19] byly v přenášených datových blocích použity pevné délky proměnných a byly im-

[19] Řazení jednotlivých bytů ve vícebytových proměnných

plementovány konverze vícebytových proměnných na formát little endian a zpět v závislosti na architektuře hostitele.

Zavedení ovladače do operačního systému probíhá automaticky po připojení převodníku k rozhraní USB počítače. Přenášet zprávy mezi počítačem a sběrnicí CAN je možné prostřednictvím znakového zařízení vytvořeného ovladačem, které využívají například programy ze skupiny CAN/CANOPEN.

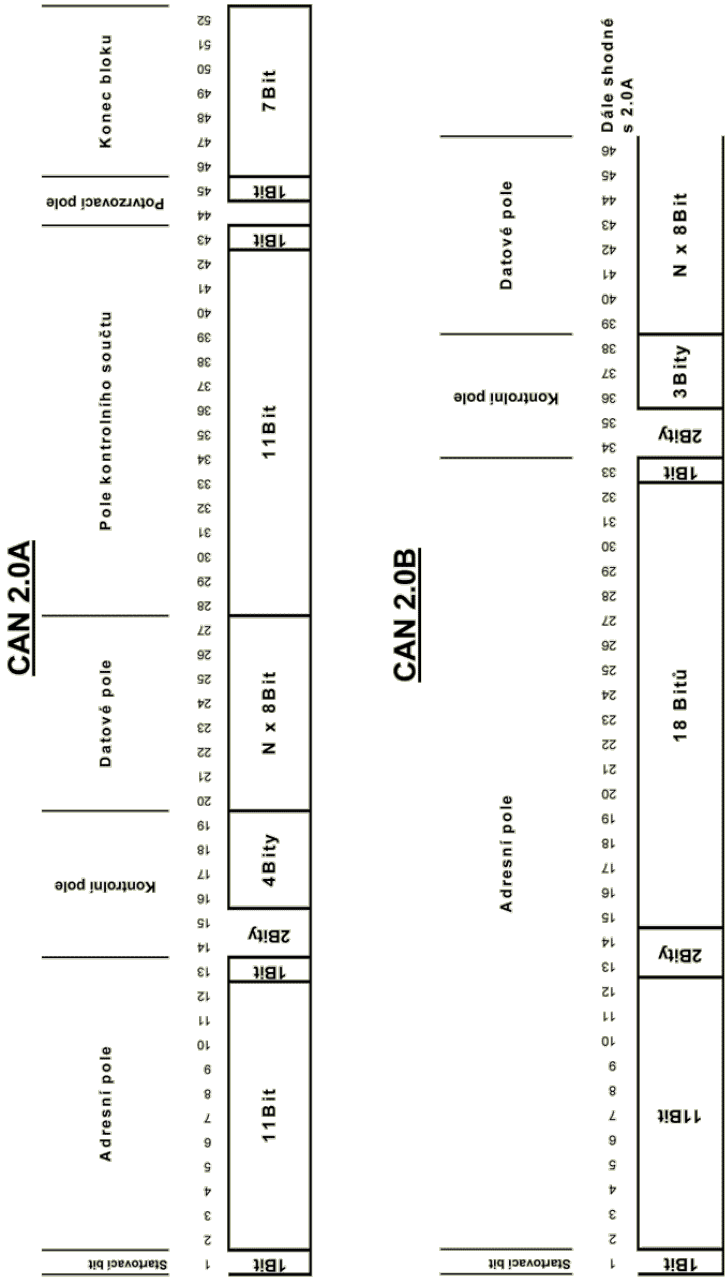
Pro testování programového řešení byl použit počítač PC s rozhraním USB a operačním systémem Linux verze jádra 2.6.23. Do připojeného převodníku byl firmware nahrán prostřednictvím sériové linky počítače, která byla dále využita pro příjem stavových a ladicích textových informací.

Během testování přenosu zpráv byl na sběrnici CAN společně s testovaným modulem přítomen přípravek, kterým bylo možné přijímat a vysílat zprávy na sběrnici CAN a zobrazovat je přes sériové rozhraní na počítači. Testy probíhaly přenosem zpráv pomocí aplikací *readburst* a *sendburst*, které jsou mezi doplňkovými programy ovladače LINCAN. Při povolení ladících výpisů z modulu převodníku bylo pozorováno zpomalení komunikace způsobené přenosem informací sériovým rozhraním.

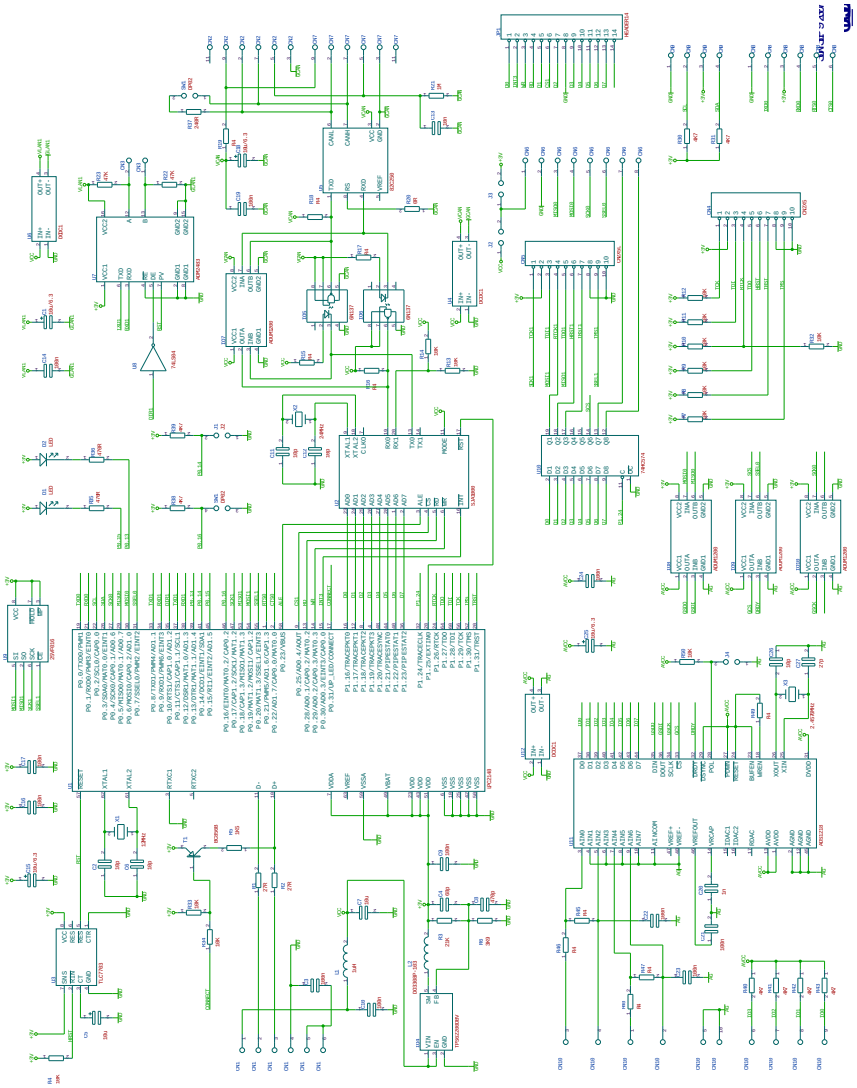
Převodník není vzhledem k vlastnostem rozhraní USB určený pro řízení procesů v reálném čase, je jím však možné řídit procesy časově nenáročné. Pro monitorování toku zpráv na sběrnici a konfigurace jednotlivých zařízení na ni připojených převodník plně dostačuje.

Zdrojové kódy modifikovaného ovladače LINCAN verze 0.3.3 a firmware řadiče implementovaného v kódu projekty uLan se nacházejí na přiloženém CD-ROM. Kód ovladače LinCAN je odzkoušen v operačním systému Linux verze jádra 2.6.23, na CD-ROM je však přiložen také patch pro jádro verze 2.6.25. Podporu převodníku USB/CAN je plánováno zařadit do hlavní vývojové větve ovladače LINCAN po sloučení s kódem aktuální CVS verze.

A Struktura datového rámce CAN [3]



B Schéma zapojení desky UL_USB1



Reference

[1] CORBET, Jonathan, RUBINI, Alessandro, KROAH-HARTMAN, Greg. Linux Device Drivers, Third edition. Andy Oram. O'Reilly : [s.n.], 2005. 615 s.
Dostupný z WWW: <http://lwn.net/Kernel/LDD3/>.

[2] ING. TARABA, Radek. Aplikování sběrnice CAN. HW.cz [online]. 2004 [cit. 2008-06-20].
Dostupný z WWW: <http://hw.cz/Rozhrani/ART1173-Aplikovani-sbernice-CAN.html>.

[3] ING. ZÁVIDČÁK, Miroslav. CAN - popis struktury. HW.cz [online]. 2004 [cit. 2008-06-20].
Dostupný z WWW: <http://hw.cz/Rozhrani/ART1111-CAN—popis-struktury.html>.

[4] Controller-area network [online]. Wikipedia.org, c2001-2008 , 25.6.2008 [cit. 2008-06-30].
Dostupný z WWW: <http://en.wikipedia.org/wiki/Controller_Area_Network>.

[5] BARTOSIŃSKI, Roman. Implementace USB interface pro počítačové periferie. Praha, 2003. 80 s. , 1 CD-ROM. Katedra řídící techniky, FEL ČVUT. Vedoucí diplomové práce Ing. Píša Pavel.
Dostupný z WWW: <http://dce.felk.cvut.cz/knihovna/diplomky/2003/>.

[6] Universal Serial Bus [online]. Wikipedia.org, c2001-2008 , 31.6.2008 [cit. 2008-06-31].
Dostupný z WWW: <http://en.wikipedia.org/wiki/Universal_Serial_Bus>.

[7] USB Implementers Forum, Inc.. Universal Serial Bus Specification : Revision 2.0. [s.l.] : [s.n.], 2000. 622 s.
Dostupný z WWW: <http://www.usb.org/>.

[8] Balanced line [online]. Wikipedia.org, c2001-2008 , 10.6.2008 [cit. 2008-06-30].
Dostupný z WWW: <http://en.wikipedia.org/wiki/Balanced_line>.

[9] Differential signaling [online]. Wikipedia.org, c2001-2008 , 9.6.2008 [cit. 2008-06-30].
Dostupný z WWW: <http://en.wikipedia.org/wiki/Differential_signaling>.

[10] ARM's way. Electronicsweekly.com [online]. 1998 [cit. 2008-06-30].
Dostupný z WWW:
<http://www.electronicsweekly.com/Articles/1998/04/29/7242/arms-way.htm>.

[11] ARM Limited [online]. Wikipedia.org, c2001-2008 , 26.6.2008 [cit. 2008-06-30].
Dostupný z WWW: <http://en.wikipedia.org/wiki/ARM_Limited>.

[12] ARM architecture [online]. Wikipedia.org, c2001-2008 , 30.6.2008 [cit. 2008-06-30].
Dostupný z WWW: <http://en.wikipedia.org/wiki/ARM_architecture>.

[13] ING. PÍŠA, Pavel, WESTENBERG, Arnaud, MOTYLEWSKI, Tomasz. Linux/RT-Linux CAN Driver (LinCAN). [s.l.] : [s.n.], 2005. 82 s.
Dostupný z WWW: <https://sourceforge.net/projects/ocera/>.

[14] UM10139 : User manual LPC214x. [s.l.] : Philips Electronics N.V., 2006. 354 s.
Dostupný z WWW: <www.nxp.com>.

[15] SJA1000 : Stand-alone CAN controller. [s.l.] : Philips Electronics N.V., 2000. 67 s.
Dostupný z WWW: <www.nxp.com>.