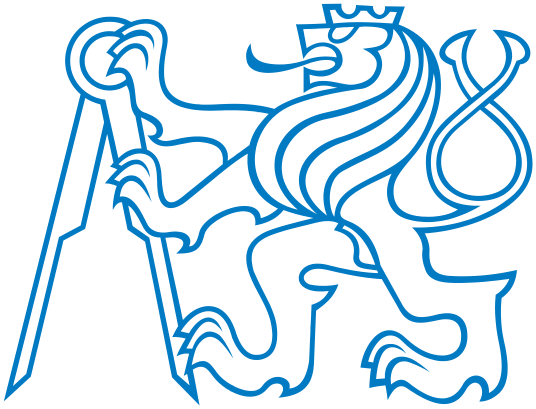
České vysoké učení technické v Praze
Fakulta elektrotechnická



Implementace driveru LinCAN pro PowerPC a porovnání se systémem Socket CAN

Diplomová Práce

Praha, 2010 **Autor: Bc. Martin Petera**

Prohlášení

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

V Praze dne 5. 1. 2010

Vaclav Petera

podpis

Poděkování

Rád bych na tomto místě poděkoval všem, kteří mi pomáhali a podporovali mě při psání práce. V prvé řadě bych rád poděkoval vedoucímu této práce Ing. Ondřeji Špinkovi, který mi poskytl nejen technické zázemí a cenné rady, ale hlavně za trpělivost a důvěru ve mne vloženou. Speciální poděkování patří také Ing. Pavlovi Píšovi za pomoc nejen s řešením problémů a otázek s jádrem driveru LinCAN. V neposlední řadě patří poděkování také mé rodině a přátelům za morální podporu a poskytnuté zázemí.

Abstrakt

V této diplomové práci je popsán postup implementace driveru LinCAN pro řadič MSCAN komunikující po CAN sběrnici a jeho srovnání s implementací SocketCAN, začleněnou do hlavní větve Vanilla kernelu. CAN řadič je součástí čipu s procesorem MPC5200/B (architektura PowerPC, výrobce Freescale Semiconductor), který je umístěn na procesorové desce nesoucí označení SHARK (výrobce Mikroklima s.r.o., divize MIDAM). Tato procesorová deska se používá při výuce a v několika projektech na Katedře Řídící Techniky Fakulty Elektrotechnické, ČVUT Praha.

Dále práce obsahuje popis a výsledky srovnávacích testů zaměřených na spolehlivost přenosu a počet přenášených zpráv. Testy jsou provedeny na standardním jádře 2.6.30 upraveném pro procesovou desku SHARK a na jádře verze 2.6.29 s aplikovaným Real-Time patchem. Část testů proběhla s deskou vstupů a výstupů dodanou spolu s procesorovým modulem vyvinutou společností Mikroklima s.r.o. Pro zbylé testy byla použita deska vstupů a výstupů vyvinutá na Katedře řídicí techniky nesoucí označení RYU.

Abstract

This diploma thesis describes implementation of the LinCAN driver for the Freescale (formerly Motorola) Scalable Controller Area Network (MSCAN) and its comparision with the SocketCAN implementation, which is integrated into the main branch of the Linux Vanilla Kernel. The CAN controller is part of the processor MPC5200/B (PowerPC architecture, Freescale Semiconductor), which is used as a main processor unit of a processor board called SHARK (developed by Mikroklima s.r.o, MIDAM division). This processor board is used in projects of Department of Control Engineering, Faculty of Eletrical engineering, Czech Technical University in Prague.

The second part of this document describes series of test and their results focused on a reliability of transmission and CAN frame transfer rate. The tests are accomplished with the standard linux kernel 2.6.30 ported for the processor board SHARK and with the linux kernel 2.6.29 including a Real-Time patch. The part of the tests passed on a input-output board shipped with SHARK proccesor module developed by Mikroklima s.r.o. For the rest of the tests was used input-output board developed at the Department of Control Engineering formerly named RYU board.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Petera**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu:

Implementace driveru LinCAN pro PowerPC a porovnání se systémem Socket CAN

Pokyny pro vypracování:

1. Seznamte se se systémem ovladačů CAN adaptérů LinCAN pro Linux a s kontrolérem sběrnice CAN v PowerPC 5200
2. Implementujte Linuxový ovladač zmíněného kontroléru a začleňte jej do systému LinCAN
3. Proveďte srovnání s existujícím systémem Socket CAN, zaměřte se hlavně na latenci (zpoždění), determinismus a spolehlivost (ztráty zpráv při vysokém zatížení sběrnice)

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Ondřej Špinka

Platnost zadání: do konce letního semestru 2010/2011

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 21.9.2009

Obsah

Seznam obrázků **ix**

Seznam tabulek **x**

1 Úvod **1**

2 CAN - Controller Area Network **2**

2.1 Popis sběrnice 2

2.2 Fyzická vrstva 3

2.2.1 Detekce chyb 3

2.2.1.1 Monitoring 3

2.2.1.2 Cyclic Redundancy Check (CRC) 4

2.2.1.3 Bit Stuffing 4

2.2.1.4 Message Frame Check 4

2.3 Zprávy 5

2.3.1 Typy rámců 5

2.3.1.1 Data Frame 5

2.3.1.2 Remote Request Frame 5

2.3.1.3 Error Frame 6

2.3.1.4 Overload Frame 6

2.3.2 Identifikátory zprávy 7

2.3.2.1 Zpráva se standardním identifikátorem 8

2.3.2.2 Zpráva s rozšířeným identifikátorem 9

2.3.3 Odeslání zprávy 10

2.3.4 Přijetí zprávy 10

3 Popis softwarových komponent **11**

3.1 Driver LinCAN 11

3.2 Driver SocketCAN 12

4 Popis použitého Hardware 14

4.1 Freescale MPC5200 14

4.1.1 Řadič MSCAN 15

4.2 Průmyslový počítač PC104 17

4.2.1 Řadič CAN SJA1000 18

4.3 CAN Analyzer KVASER (PCMCIA) 18

5 Vývoj ovladače LinCAN pro řadič MSCAN 19

5.1 Příprava HW 19

5.2 Příprava vývojového a testovacího prostředí 20

5.2.1 Kompilace pro cílovou architekturu PowerPC 20

5.2.2 Cross-kompilace jádra 20

5.2.3 Spuštění operačního systému pro MPC5200 21

5.3 Popis implementace ovladače LinCAN 21

5.4 Vybrané části implementace 23

5.4.1 Získání systémových prostředků 23

5.4.2 Obsluha komunikace po sběrnici 24

5.4.2.1 Nastavení komunikační rychlosti 24

5.4.2.2 Nastavení filtru přijímaných zpráv 24

5.4.2.3 Příjem zpráv 24

5.4.2.4 Odesílání zpráv 24

5.4.2.5 Obsluha a maskování přerušení 25

6 Srovnání ovladačů SocketCAN a LinCAN 26

6.1 Vlastnosti obou implementací 26

6.2 Metody srovnání 26

6.2.1 Testovací aplikace Canburst 27

6.3 Varianty testů 28

6.3.1 Propustnost při různých nastavení bitrate 28

6.3.2 Propustnost zatížené sběrnice 31

6.3.3 Připojení více aplikací na jeden komunikační objekt 31

6.3.4 Propustnost při užití obou řadičů MSCAN 32

6.3.4.1 Oba řadiče MSCAN ve funkci vysílače 33

6.3.4.2 Řadiče MSCAN ve funkci vysílače i posluchače 34

6.3.4.3 Oba řadiče MSCAN ve funkci posluchače 35
6.3.5 Spolehlivost při velkém počtu přenášených zpráv 35

7 Závěr **37**

Literatura **38**

A Naměřená data - bitrate **I**
A.1 Bitrate 50 kbps I
A.1.1 Odesílání I
A.1.2 Příjem II
A.2 Bitrate 125 kbps III
A.2.1 Odesílání III
A.2.2 Příjem IV
A.3 Bitrate 250 kbps V
A.3.1 Odesílání V
A.3.2 Příjem VI
A.4 Bitrate 500 kbps VII
A.4.1 Odesílání VII
A.4.2 Příjem VIII
A.5 Bitrate 1000 kbps IX
A.5.1 Odesílání IX
A.5.2 Příjem X

B Mapování registrů řadiče MSCAN **XII**

C Obsah přiloženého CD **XIV**

Seznam obrázků

2.1 Schéma sběrnice podle ISO 11898 3
2.2 Bit Stuffing 4
2.3 Standardní Data/Remote Frame 8
2.4 Rozšířený Data/Remote Frame 9

3.1 Ilustrace socketové vrstvy 13

4.1 Procesor MPC5200 Shark DB1 s daughterboardem 14
4.2 Schéma filtrování zpráv užitím 16-bitových filtrů 16
4.3 Ilustrační foto průmyslového počítače PC104 17
4.4 Ilustrační foto PCMCIA karty CAN Analyzeru KVASER 18

6.1 Blokové schéma zapojení pro měření bitrate 28
6.2 Závislost teoretické a skutečné propustnosti - příjem, 1000 kbps 28
6.3 Závislost teoretické a skutečné propustnosti - odesílání, 50 kbps 29
6.4 Závislost teoretické a skutečné propustnosti - odesílání, 125 kbps 29
6.5 Závislost teoretické a skutečné propustnosti - odesílání, 250 kbps 30
6.6 Závislost teoretické a skutečné propustnosti - odesílání, 500 kbps 30
6.7 Závislost teoretické a skutečné propustnosti - odesílání, 1000 kbps 30
6.8 Blokové schéma zapojení pro měření zatížené sběrnice 31
6.9 Blokové schéma zapojení pro měření při více klientských aplikacích . . . 32
6.10 Blokové schéma zapojení pro měření při použití obou řadičů MSCAN . . 32
6.11 Blokové schéma zapojení pro měření výpadku zpráv 36

Seznam tabulek

2.1 Popis jednotlivých částí zprávy se standardním identifikátorem 8

5.1 Implementace ovladače - obsah zdrojových kódů 22
5.2 Implementace ovladače - modifikace LinCAN frameworku 23

6.1 Canburst - Použitá makra pro řízení překladu 27
6.2 Tabulka hodnot - oba řadiče MSCAN ve funkci vysílače, jádro 2.6.30 . . 33
6.3 Tabulka hodnot - oba řadiče MSCAN ve funkci vysílače, jádro 2.6.29-rt . 33
6.4 Tabulka hodnot - řadiče MSCAN jako vysilač i posluchač, jádro 2.6.30 . 34
6.5 Tabulka hodnot - řadiče MSCAN jako vysilač i posluchač, jádro 2.6.29-rt 34
6.6 Tabulka hodnot - oba řadiče MSCAN ve funkci posluchače, jádro 2.6.30 . 35
6.7 Tabulka hodnot - oba řadiče MSCAN ve funkci posluchače, jádro 2.6.29-rt 35
6.8 Tabulka hodnot - test spolehlivosti 36

A.1 Tabulka hodnot - odesílání, 50 kbps, jádro 2.6.30 I
A.2 Tabulka hodnot - odesílání, 50 kbps, jádro 2.6.29-rt II
A.3 Tabulka hodnot - příjem, 50 kbps, jádro 2.6.30 II
A.4 Tabulka hodnot - příjem, 50 kbps, jádro 2.6.29-rt III
A.5 Tabulka hodnot - odesílání, 125 kbps, jádro 2.6.30 III
A.6 Tabulka hodnot - odesílání, 125 kbps, jádro 2.6.29-rt IV
A.7 Tabulka hodnot - příjem, 125 kbps, jádro 2.6.30 IV
A.8 Tabulka hodnot - příjem, 125 kbps, jádro 2.6.29-rt V
A.9 Tabulka hodnot - odesílání, 250 kbps, jádro 2.6.30 V
A.10 Tabulka hodnot - odesílání, 250 kbps, jádro 2.6.29-rt VI
A.11 Tabulka hodnot - příjem, 250 kbps, jádro 2.6.30 VI
A.12 Tabulka hodnot - příjem, 250 kbps, jádro 2.6.29-rt VII
A.13 Tabulka hodnot - odesílání, 500 kbps, jádro 2.6.30 VII
A.14 Tabulka hodnot - odesílání, 500 kbps, jádro 2.6.29-rt VIII

A.15 Tabulka hodnot - příjem, 500 kbps, jádro 2.6.30 VIII
A.16 Tabulka hodnot - příjem, 500 kbps, jádro 2.6.29-rt IX
A.17 Tabulka hodnot - odesílání, 1000 kbps, jádro 2.6.30 IX
A.18 Tabulka hodnot - odesílání, 1000 kbps, jádro 2.6.29-rt X
A.19 Tabulka hodnot - příjem, 1000 kbps, jádro 2.6.30 X
A.20 Tabulka hodnot - příjem, 1000 kbps, jádro 2.6.29-rt XI

B.1 Mapování základních registrů řadiče MSCAN do paměti XII
B.2 Mapování registrů filtru příchozích zpráv řadiče MSCAN do paměti . . . XIII
B.3 Mapování registrů RX a TX bufferů řadiče MSCAN do paměti XIII

Kapitola 1

Úvod

Sběrnice CAN - *Controller Area Network* [1], definovaná *normou ISO 11898 a CAN specification 2.0A/B* [2], je v dnešní době hojně využívána v automobilovém průmyslu. Dále se používá i v odvětvích, kde je zapotřebí jednoduché ale přesto robustní komunikační řešení nebo kde je kladen důraz na garantovanou dobu přenosu i při větším počtu zařízení na sběrnici. CAN je běžně dostupná a cenově přijatelná technologie, což ji staví na pozici jedné z nejpoužívanějších průmyslových sběrnic. V dnešní době jsou běžně řadiče sběrnice CAN zabudovány jako součást v mikrokontrolerech a embedded zařízeních.

Jedním ze systémů, kde je zabudován řadič CAN, je i procesor Freescale MPC5200B [3] osazen na základní desce SHARK. Tento systém je používán pří výuce i v projektech na Katedře řídící techniky Fakulty Elektrotechnické, Českého Vysokého Učení Technického v Praze. Tato práce se zabývá návrhem a implementací ovladače pro operační systémy Linux a Real-Time Linux v rámci projektu LinCAN (viz. [4]) a jeho srovnáním s řešením SocketCAN [5]. LinCAN je ovladač pro řadiče CAN vyvíjený jako součást projektu OCERA [6]. SocketCAN je podpora CAN řadičů využívající socketovou infrastrukturu jádra systému Linux [7].

První část práce obsahuje popis CAN sběrnice a její fyzické vrstvy, stručný úvod k driveru LinCAN, popis systému SocketCAN, seznámení s hardwarem použitým při tvorbě a verifikaci funkčnosti ovladače. V druhé části je popsán vývoj ovladače LinCAN, návrh struktury a některá úskalí specifická pro použitý řadič MSCAN. Poslední část pak obsahuje popis a výsledky testů, které slouží k porovnání obou ovladačů z hlediska spolehlivosti a počtu přenesených zpráv za jednotku času.

Kapitola 2

CAN - Controller Area Network

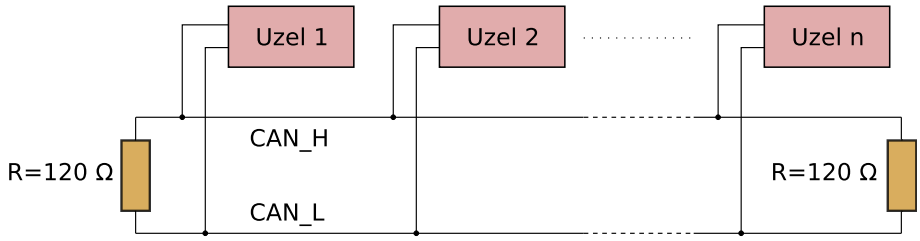
2.1 Popis sběrnice

CAN je průmyslová seriová multimaster sběrnice s rychlostí přenosu až 1 Mb/s při maximální délce cca 50 m. Zařízení nemají žádnou vlastní adresu, proto je možné je přidávat, aniž by byl nutný zásah do softwaru ostatních zařízení na sběrnici. Komunikace probíhá pomocí zpráv obsahujících identifikátor, podle kterého podřízené uzly rozhodují, zda zprávu přijmou či nikoliv. Jelikož zpráva neobsahuje adresu příjemce, ale pouze identifikátor zprávy, je možné, aby vyslaná zpráva byla přijata všemi zařízeními zároveň nebo naopak nebyla přijata žádným z nich. Sběrnice aplikuje prioritizaci zpráv na základě identifikátoru zprávy. CAN je Fault-tolerant, což znamená, že během přenosu dochází k detekci chyb. Při zjištění chyby při přenosu je zpráva znovu odeslána a zvýší se hodnota počítadla chyb. Při dosažení určité hranice počtu chyb přejde zařízení do *Error Passive* režimu, což znamená, že zařízení se nadále účastní výměny zpráv, ale při detekci chyby vysílá *Error Passive Flag* (viz. 2.3.1.3). Pokud uzel posílá velké množství chybných zpráv, dojde k jeho odpojení od sběrnice (přechod do stavu *Bus Off*) a tím k zajištění průchodnosti sběrnice.

Sběrnici CAN začala v roce 1983 vyvíjet společnost Robert Bosch GmbH, aby ji v roce 1986 oficiálně uvedla na trh. Následně v roce 1987 byly vyvinuty čipy pro CAN společnostmi Intel a Philips Semiconductors. V roce 1993 vydala firma Bosch specifikaci CAN 2.0 [2] a v témže roce vznikl standard ISO 11898, který byl v roce 1995 rozšířen o *Extended Frame* formát. Tento rošířený formát rozšiřuje sadu použitelných identifikátorů zprávy z původních 2048 na 2^{29} (přes 500 milionů) možných identifikátorů zprávy.

2.2 Fyzická vrstva

Fyzická vrstva může být tvořena různými medii (dvouvodičová linka, optické vlákno apod.). Základním požadavkem na medium je, aby realizovalo logický součin (kvůli prioritizaci zpráv - 2.3.3). Nejčastěji se jako přenosové medium používá rozdílová linka (ISO 11898) se zakončovacími rezistory. Sběrnice se skládá ze dvou vodičů (CAN_H a CAN_L), dominantní a recesivní logická úroveň se rozlišuje podle rozdílového napětí těchto vodičů (recesivní úroveň $U_{diff} = 2$ V, dominantní úroveň $U_{diff} = 0$ V). Sběrnice je proto zakončena na obou koncích pull-up rezistory o velikosti R = 120 Ω, které zároveň zabraňují odrazům na koncích vedení.



Obrázek 2.1: Schéma sběrnice podle ISO 11898

2.2.1 Detekce chyb

2.2.1.1 Monitoring

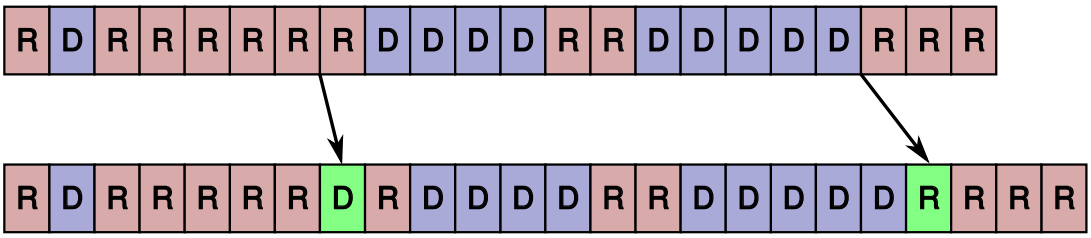
Každý vysílající uzel zároveň sleduje logickou úroveň na sběrnici. Pokud detekuje jinou log. úroveň než vysílá, ukončí vysílání a až do konce přenosu zprávy působí jako posluchač. Tento případ může nastat pouze v případě, že jeden uzel vysílá recesivní úroveň a jiný uzel vysílá dominantní úroveň. Této vlastnosti se využívá během arbitrace pro prioritizaci zpráv, zpráva s nižším číslem identifikátoru obsahuje více dominantních bitů.

2.2.1.2 Cyclic Redundancy Check (CRC)

Každá zpráva obsahuje na svém konci CRC kontrolní součet generovaný z hodnot polí *Start of Frame*, *Arbitration Field*, *Control Field* a *Data Field*. CRC se získá jako zbytek po dělení posloupnosti bitů ve zmíněných polích polynomem $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Kontrolní součet používají posluchači k ověření integrity přijatých dat. Při nekonzistenci kontrolního součtu a obsahu zprávy neodešle posluchač *ACK* bit v dominantní úrovni, čímž nepotvrdí přijetí neporušené zprávy.

2.2.1.3 Bit Stuffing

Jelikož není přenášen hodinový signál, používá se *Bit stuffing* jako způsob dodržení synchronizace jednotlivých uzlů na sběrnici. Postup kódování je jednoduchý, za každou sekvenci pěti po sobě následujících bitů stejné logické úrovně je vložen jeden bit opačné log. úrovně. Toto kódování se používá pro pole *Arbitration Field*, *Control Field*, *Data Field* a *CRC Field* u zpráv typu *Data Frame* a *Remote Frame Request*. U zpráv *Error Frame* a *Overload Frame* se nepoužívá, protože tyto zprávy mají fixní délku.



Obrázek 2.2: Bit Stuffing

2.2.1.4 Message Frame Check

Message Frame Check provádí kontrolu bitů v místech pevně daného formátu zprávy (například CRC Delimiter, nebo rezervované bity v *Control Filed*), aby odpovídaly definici ve standardu CAN [2].

2.3 Zprávy

Komunikace po sběrnici probíhá pomocí zpráv (rámců). Rozlišují se 4 typy zpráv: *Data Frame (viz. 2.3.1.1)*, *Remote Request Frame (viz. 2.3.1.2)*, *Error Frame (viz. 2.3.1.3)* a *Overload Frame (viz. 2.3.1.4)*. *Data Frame* a *Remote Request Frame* obsahují identikátor zprávy (viz. 2.3.2), který slouží k arbitraci přístupu ke sběrnici a ostatním (podřízeným) zařízením k rozhodování, zda přijmout či nepřijmout vysílanou zprávu. Oproti tomu *Error Frame* a *Overload Frame* identifikátory neobsahují a neslouží primárně k výměně dat. Každý typ rámce má svůj specifický účel a použití.

2.3.1 Typy rámců

2.3.1.1 Data Frame

Tento rámec (typ zprávy) se používá pro přenos dat mezi jednotlivými uzly. Skládá se ze sedmi částí (*Start Of Frame*, *Arbitration Field*, *Control Field*, *Data Field*, *CRC Field*, *Acknowledge Field* a *End Of Frame* (viz. 2.3.2). Jednotlivé rámce jsou při vysílání po sběrnici odděleny sekvencí bitů *Interframe Space*, která má délku minimálně tři bity. *Interframe Space* je okamžik, kdy je sběrnice volná a tedy každý uzel může zahájit vysílání odesláním dominantního bitu *Start Of Frame*. Maximální počet datových bytů je osm, minimální nula. Přenos zprávy s nulovým počtem datových bytů je rychlý neboť zpráva má malou délku (ve standardním formátu pouze 37 bitů). Taková datová zpráva se proto používá pro rychlé přenosy, kde se využívá jako nositele informace jejího identifikátoru.

2.3.1.2 Remote Request Frame

Rámec se používá při komunikaci jako žádost o data. Například pokud se jedná o centralizovaný systém, může zařízení, které zpracovává údaje od měřicích jednotek s CAN rozhraním, použít tento rámec, aby vyzval ostatní zařízení k odeslání dat. Některé CAN řadiče podporují automatické odeslání *Data Frame* při přijetí *Remote Request Frame* bez nutnosti obsluhy softwarem. *Remote Request Frame* je strukturou podobný jako *Data Frame*, liší se bitem RTR (*Remote Transfer Request*), který má u *Remote Request Frame* recesivní úroveň, dále neobsahuje tento rámec část *Data Field*, ikdyž v části *Control Field* může být hodnota DLC (*Data Length Code*) nenulová, v tomto případě údáva hodnota počet očekávaných datových bytů.

2.3.1.3 Error Frame

Error Frame je odvysílán uzlem, který detekoval chybu při přenosu. Chybou může být například porušení pravidla *Bit Stuffing - 2.2.1.3*. Rámec *Error Frame* se skládá ze dvou částí: *Error Flag* a *Error Delimiter*. Error flag je šest po sobě následujících bitů - buď recesivní úrovně (*Error Passive Flag*), vysílá-li je zařízení v *Error Passive* módu, nebo dominantní úrovně (*Error Active Flag*), pokud zařízení, které detekovalo chybu je v *Error Active* módu. Při vysílání *Error Flag* není aplikováno pravidlo *Bit Stuffing*, což detekují ostatní zařízení na sběrnici jako chybu. Po odvysílání prvních šesti bitů - *Error Flag* - odešle zařízení bit recesivní úrovně a čeká, než se na sběrnici objeví recesivní úroveň. Poté dokončí přenos *Error Frame* odesláním sedmi bitu recesivní úrovně.

2.3.1.4 Overload Frame

Overload Frame je zpravidla vysílán posluchačem při žádosti o prodlevu mezi jednotlivými datovými rámci. Rámec *Overload Frame* se skládá ze dvou částí zvaných *Overload Flag* a *Overload Delimiter*. Overload flag je šest po sobě následujícíh bitů dominantní úrovně (stejně jako *Error Active Flag* v *Error Frame*). Po prvních šesti bitech následuje, podobně jako u *Error Frame*, odeslání recesivního bitu a čekaní na hranu z dominantní na recesivní úroveň. Po detekování této hrany odvysílá každý uzel sedm recesivních bitů a tím ukončí *Overload Frame*. Mezi *Data* a *Remote Frame* mohou být odeslány nejvýše dva *Overload Frame* a to v následujících případech:

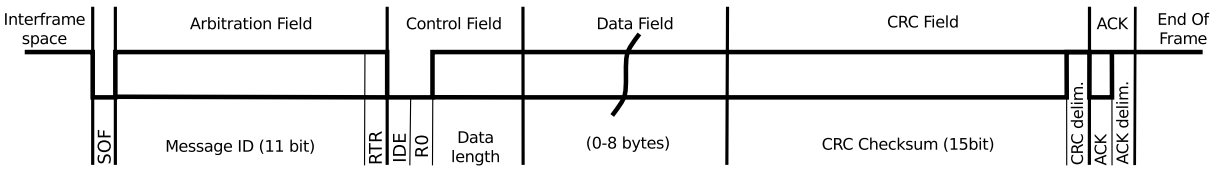
- Uzel, který přijímal zprávu, žádá o krátkou prodlevu před vysíláním dalšího *Data* nebo *Remote Frame*.
- Při detekci dominantních bitů na prvním a druhém místě *Interframe Space* (prostor mezi *Data* a *Remote Frame*).
- Pokud uzel zjistí dominantní úroveň posledního bitu části *Error* nebo *Overload Delimiter*.

2.3.2 Identifikátory zprávy

CAN sběrnice používá u zprávy místo adresy příjmce číselný identifikátor, který vyjadřuje charakter zprávy. Jaké číslo identifikátoru odpovídá konkrétnímu typu zprávy je stanoveno aplikační úrovní. Zprávy se vysílají metodou broadcasting, existuje tedy jeden vysílající a více přijímacích uzlů. Každý přijímací uzel se podle čísla identifikátoru zprávy rozhodne, zda zprávu přijme (viz. 2.3.4). V *Data a Remote Request Frame* se používají dva typy identifikátorů, *standardní* a *rozšířený identifikátor*.

2.3.2.1 Zpráva se standardním identifikátorem

Standardní indetifikátor je součástí již první verze CAN specifikace. Kvůli zpětné kompatibilitě se používá i po zavedení rozšířeného identifikátoru. Délka *standardního identifikátoru* je 11 bitů, což dává prostor pro 2048 různých typů zpráv. Identifikátor se odesílá jako součást *Arbitration Field* a využívá se jí k prioritizaci přístupu ke sběrnici.



Obrázek 2.3: Standardní Data/Remote Frame

název	význam
Start of Frame	Začátek zprávy je vždy vysílán jako dominantní úroveň.[1]
Arbitration Field	Obsahuje identifikátor zprávy (viz. 2.3.2) a příznak RTR (Remote Transfer Request). RTR odlišuje *Data Frame* od *Remote Frame*.
Control Field	První bit nese informaci zdali se jedná o *Standard* nebo *Extended Frame* (dominantní pro *Standard Frame*), druhý bit je rezervovaný, poté nasledují 4 bity nesoucí informaci o počtu datových bytů.
Data Field	Obsahuje 0 až 8 datových bytů.
CRC Field	Obsahuje 15 bitů CRC kontrolního součtu a CRC Delimiter (slouží k lepšímu oddělení prvního ACK bitu od *CRC Field*).
ACK Field	Je potvrzení přijetí nepoškozené zprávy a skládá ze ze dvou bitů.[2]
End of Frame	Indikuje konec zprávy a je vysílán jako 7 recesivních bitů.

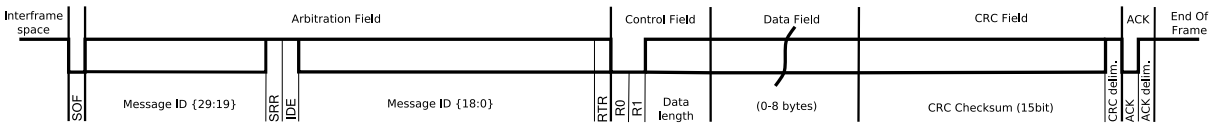
Tabulka 2.1: Popis jednotlivých částí zprávy se standardním identifikátorem

[1] Před vysláním *Start of Frame* čeká *master* minimálně na tři bity recesivní úrovně, pak zahájí vysílání. Uzel, který vysílal předchozí *Data Frame* nebo *Remote Frame*, čeká osm bitů před zahájením vysílání. Umožní tak ostatním uzlům zahájit vysílání dříve.

[2] *Master* vysílá oba bity na recesivní úrovni, *slave*, pokud přijatý CRC součet odpovídá vypočítanému, vyšle první bit dominantní a druhý recesivní, čímž získá *master* potvrzení o doručení. Tato část nese

2.3.2.2 Zpráva s rozšířeným identifikátorem

V roce 1995 přibyl do specifikace ISO 11898 rozšířený formát zprávy. Identifikátor je v této zprávě reprezentován 29 bity, což umožňuje použít přes půl miliardy různých zpráv. Význam horních 11 bitů identifikátoru je shodný s významem identifikátoru standardní zprávy, zbývajících 18 bitů je použito jako upřesňující rozlišení zprávy.



Obrázek 2.4: Rozšířený Data/Remote Frame

Identifikátor je v rozšířeném typu zprávy rozdělen kvůli kompatibilitě se standardním formátem na dvě části. Po odeslání *Start Of Frame (SOF)* se odesílá horních 11 bitů identifikátoru počínaje MSB. Následující bit (ve standardní zprávě *Remote Transfer Request (RTR)*) je v rozšířeném formátu nahrazen *SRR* (*Substitude Remote Request*), který je vždy recesivní úrovně. Za *SRR* následuje *Identifier Extension IDE*, který rozlišuje standardní a rozšířený formát, u rozšířeného formátu je *recesivní*. Ve standardním formátu zprávy je *IDE* bit již součástí *Control Filed*, zde je ještě součástí *Arbitration Field*. Za *IDE* bitem následuje zbývajících 18 bitů identifikátoru a *RTR* bit, který ukončuje *Arbitration Field*. Zbytek rozšířené zprávy se shoduje se standardní zprávou, pouze první bit *Control Field* nemá význam *IDE*, ale je rezervovaný.

pouze informaci zdali *slave* přijal zprávu neporušenou, nevypovídá nic o procesu filtrace zprávy pomocí LAF (*Local Acceptance Filter* - 2.3.4).

2.3.3 Odeslání zprávy

Pro arbitraci je použito pravidla *Monitoringu* (viz. 2.2.1.1). Jeden uzel přestane vysílat, jakmile detekuje při vyslání recesivní úrovně na sběrnici úroveň dominantní. Druhý uzel pokračuje bez povšimnutí dále v procesu přenosu zprávy. Takto na konci arbitrace vysílá pouze jeden uzel. V případě, kdy se ve stejný okamžik snaží dva uzly o vyslání zprávy se stejným identifikátorem, je toto detekováno stejným principem v sekci *Control Field* např. odlišnou délkou dat nebo přímo v *Data Field*. Jestliže dvě zařízení vysílají zprávu se stejným identifikátorem i daty, není to detekováno jako chyba.

2.3.4 Přijetí zprávy

Každý uzel, který nevysílá, je automaticky příjemce a čte bity ze sběrnice. Po přijetí *CRC Field* překontroluje výsledný kontrolní součet zprávy. Pokud souhlasí s přijatým CRC, potvrdí příjem dominantním prvním bitem *ACK Field*. Teprve potom, co zpráva byla vyhodnocena jako neporušená, přistupuje zařízení k vyhodnocení žádanosti zprávy. Postupně porovná identifikátor zprávy s identifikátorem a bitovou maskou přijímacího filtru. Pokud najde shodu, umístí zprávu do *mailboxu* a provede příslušné kroky k oznámení přijaté zprávy (zpravidla vyvolá přerušení). Některé řadiče (např. *HCAN2*) jsou vybaveny více přijímacími mailboxy, identifikátor zprávy může odpovídat nastavené masce u více mailboxů, záleží pak na konkrétní hardwarové implementaci, zdali se zpráva umístí jen do prvního či do všech odpovídajících mailboxů.

Kapitola 3

Popis softwarových komponent

3.1 Driver LinCAN

LinCAN [4] je driver pro řadiče CAN vyvíjený na Katedře řídicí techniky pro systémy Linux, RTLinux a další. Driver funguje jako modul, který se přidává do systému standardními příkazy *insmod* nebo *modprobe* a odebírá příkazem *rmmod*. Po přidání modulu je možné přistupovat k objektům jako k standardním znakovým zařízením pomocí */dev/can0, /dev/can1* atd.

Ovladač LinCAN byl z počátku vyvíjen pro *ISA sběrnici*, která používá pevné adresy zařízení. Proto LinCAN plně nepodporuje technologii *Plug and Play* a nespolupracuje s *udev*, proto je nutné přidat *device nody* do adresáře */dev/* ručně a to pomocí příkazu *mknod*. K takto vytvořeným *device nodům* je pak možné přistupovat pomocí systémových volání *open*, *close*, odesílat a příjímat zprávy pomocí *read*, *write* a nastavovat *Local Acceptance Mask*, případně rychlost komunikace, voláním *ioctl*. Pro výměnu CAN zpráv mezi uživatelskou aplikací a driverem slouží struktura *canmsg_t* definovaná v hlavičkovém souboru *canmsg.h* ovladače LinCAN. V tomto souboru je i definice struktury *canfilt_t*, pomocí které je možné nastavit filtr pro příjem zpráv.

Filtrování zpráv se v driveru LinCAN provádí na dvou úrovních, softwarově v jádře driveru a pokud je filtrování podporováno hardwarem, tak i přímo na řadiči. Jeden *device node* je možné otevřít vícenásobně, i když tento *device node* zpravidla reprezentuje jeden fyzický buffer sloužící k výměně zpráv. Proto je nutné pro jednotlivé konce front z uživatelských aplikací udržovat nastavení filtru a z HW vrstvy jim předávat pouze ty zprávy, které si vyžádaly. Při nastavení filtru uživatelskou aplikací se provede v ovladači

sjednocení filtrů pro příslušný komunikační objekt a výsledný filtr se propaguje do hardwarové vrstvy. V případě velkého provozu na sběrnici může hardwarové filtrování výrazně snížit zátež systému.

Při volání *write* je možné předat blok paměti obsahující více zpráv. Tyto zprávy jsou pak odeslány řadičem ihned po sobě, samozřejmě s ohledem na arbitraci na sběrnici. Vnitřní struktura driveru je navržena tak, aby splňovala požadavky na RealTime systém. Tedy striktně deterministické chování s důrazem na rychlou obsluhu přerušení a co možná nejmenší *overhead* spojený s obsluhou sběrnice.

3.2 Driver SocketCAN

Volkswagen Group Research vyvíjel od listopadu 2002 ovladač pro řadiče CAN sběrnice známý jako *Low Level CAN Framework*. Tento ovladač byl později pod názvem *SocketCAN* integrován do jádra systému Linux[1]. Driver využívá vrstvu *socketů* v jádře, použití v uživatelských aplikacích je pak velmi podobné práci s často používanými síťovými sockety. Díky socketové vrstvě umožňuje ovladač navázat pomocí systémového volání *bind* více socketů na jeden interface, což do té doby většina ovladačů CAN sběrnice využívající *znakové zařízení* neumožňovala[2]. Implementace ovladače *SocketCAN* pracuje, na rozdíl od *LinCANu*, s dynamickým přidělováním prostředků (paměť IO, přerušení) využívající například *OpenFirmware*[3]. SocketCAN v systému emuluje síťové zařízení (canX) a je možné ho ovládat pomocí příkazu *ifconfig* podobně jako například *eth0*. Výměna zpráv mezi uživatelskou aplikací využívá strukturu *can_frame*, která obsahuje identifikátor zprávy, data a příznaky jako např. CAN_EXT pro rošířený formát identifikátoru či CAN_RTR pro *Remote Request*.

Pro nadstavbu jádra *XENOMAI* existuje ovladač *RT CAN* [8], který poskytuje RealTime vlastnosti.

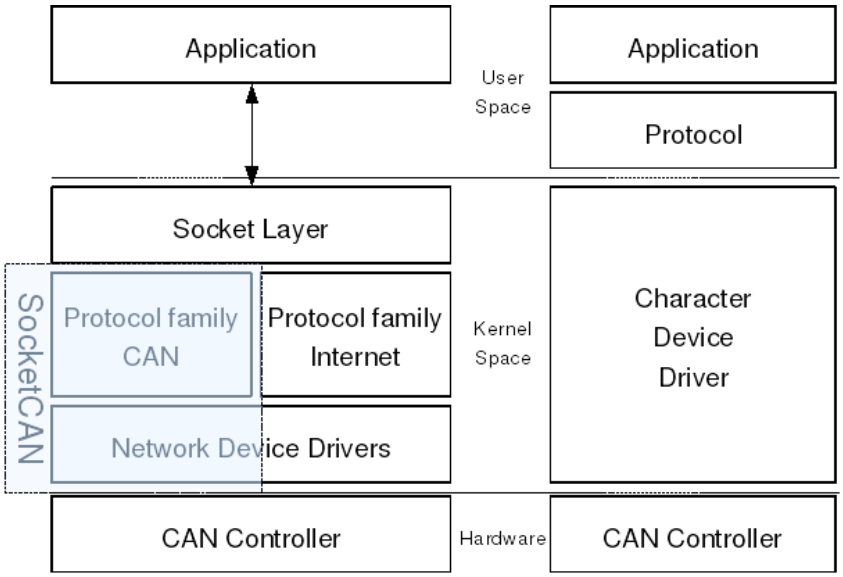
[1] poprvé se objevil ve verzi 2.6.25

[2] LinCAN toto umožňuje, ač se jedná o ovladač "schovaný" za znakové zařízení

[3] Open Firmware je interface pro získání informací o hardwaru, jako například mapování do paměťového adresního prostoru nebo přidělené přerušení.

SocketCAN zavádí do jádra novou rodinu protokolu (Protocol Family) *PF_CAN* a rodinu adres (Address Family) *AF_CAN*, které se využívají při vytváření socketu k definici přijímaných a vysílaných identifikátorů. Pro příjem zpráv s různými identifikátory je proto nutné pro každý identifikátor zprávy použít nový socket.

Filtr zpráv je možné nastavit voláním *setsockopt*. Filtrování se však provádí ve vrstvě socketů, tedy softwarově. Z toho vyplývá, že filtrování zpráv využívá systémových prostředků, i když se jedná o zprávy, které nejsou pro příslušný uzel na sběrnici vůbec určené.



Obrázek 3.1: Ilustrace socketové vrstvy

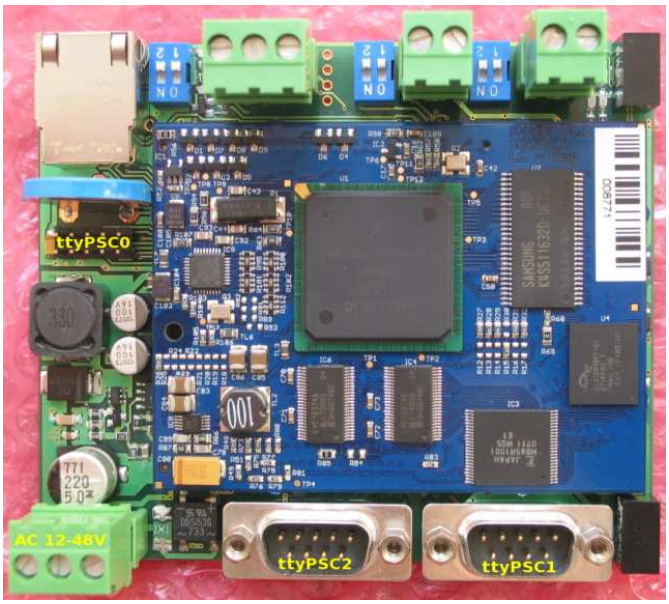
(zdroj: wikipedia.org)

Kapitola 4

Popis použitého Hardware

4.1 Freescale MPC5200

MPC5200 je procesor s jádrem e300, které vychází z architektury PowerPC. Je navržen tak, aby vyhovoval použití v mobilních zařízeních, obsahuje tedy kromě výkoného superskalárního jádra i velké množství periferií. Součástí *MPC5200* je interface pro SDRAM a DDR paměť, sběrnice PCI (ver 2.2), ATA interface (pro připojení IDE disků), Fast Ethernet, dva USB 1.1 kanály, řadič CAN, SPI, JTAG a další.



Obrázek 4.1: Procesor MPC5200 Shark DB1 s daughterboardem

(zdroj: http://rtime.felk.cvut.cz)

Procesor podporuje různě úrovně nastavení pro nízkou spotřebu (*nap, doze, sleep a deep sleep*) a zároveň *WakeUp* pomocí hodin realného času (RTC) nebo vnějších periferií (GPIO, CAN). Vlastní jádro dosahuje výpočetního výkonu až 760 MiPS při frekvenci hodin 400 MHz, pracující nad 32-bitovými adresním prostorem s 64-bitovou datovou sběrnicí, která umožňuje přenos dvou slov zároveň. Procesor MPC5200 má dvě externí sběrnice. Jednu vyhrazenou pro připojení paměti a zařízení s přímým přístupem do paměti. Druhá sběrnice slouží pro komunikaci s periferiemi, které nevyužívají *DMA*, např. GPIO, Ehternet nebo řadič MSCAN. Paměťový interface dokáže pracovat s pamětí typu SDRAM či DDR na frekvenci až 132 MHz o maximální kapacitě 512 MB na dvou čipech (omezenou dvěma signály *ChipSelect*)[1].

4.1.1 Řadič MSCAN

Procesor MPC5200 má integrované dva řadiče CAN sběrnice *Motorola Scallable Controller Area Network* (MSCAN). Řadič MSCAN odpovídá specifikaci sběrnice CAN 2.0A/B [2], tedy podporuje jak standardní, tak rozšířený identifikátor zprávy, 0 až 8 datových bitů, komunikační rychlost až 1 Mbps a Remote Frames. Pro příjem slouží čtyřslotová FIFO fronta pro zprávy, pro odesílání je možné použít 3 nezávislé TX buffery, kterým je možné přiřadit lokální priority dle kterých MSCAN zpracovává požadavky na odeslání. Při vhodném používání všech tří TX bufferů je možné odesílat neustále zprávy bez toho, aniž by prodleva mezi zprávami byla delší nez *Interframe Space* (viz. 2.3.1.1). Každý ze dvou řadičů může generovat přerušení v závislosti na nastavení (např. při príjmu zprávy, při uvolnění odesílacího bufferu a dalších událostech).

Řadič je mapován do paměti o velikosti 128 bytů, ve kterých je mapováno třicet 8-bitových registrů sloužících k nastavení a ovládání řadiče, přijímací (RX) buffer a jeden z trojice odesílacích (TX) bufferů. K nastavení základních vlastností, jako jsou jednotlivé urovně napájení (*Sleep, Power Down a CAN Disabled*), *WakeUp* při aktivitě sběrnice a žádosti a potvrzení o přechod do režimu konfigurace či spánku, slouží dva řídicí registry. Nastavení komunikační rychlosti se provádí pomocí dvou registrů, ve kterých jsou zakódovány časovací údaje (*Time Segmenty 1 a 2*, Synchronization Jump Width a počet vzorků na bit). Dále jsou v paměti mapovány registry obsahující stav *Error counterů*, příznaky a masky přerušení pro příjem i vysílání. Mapovány jsou i registry sloužící k výběru odesílacího bufferu (viz. 5.4.2.4) a registry pro žádost o zrušení vysílání

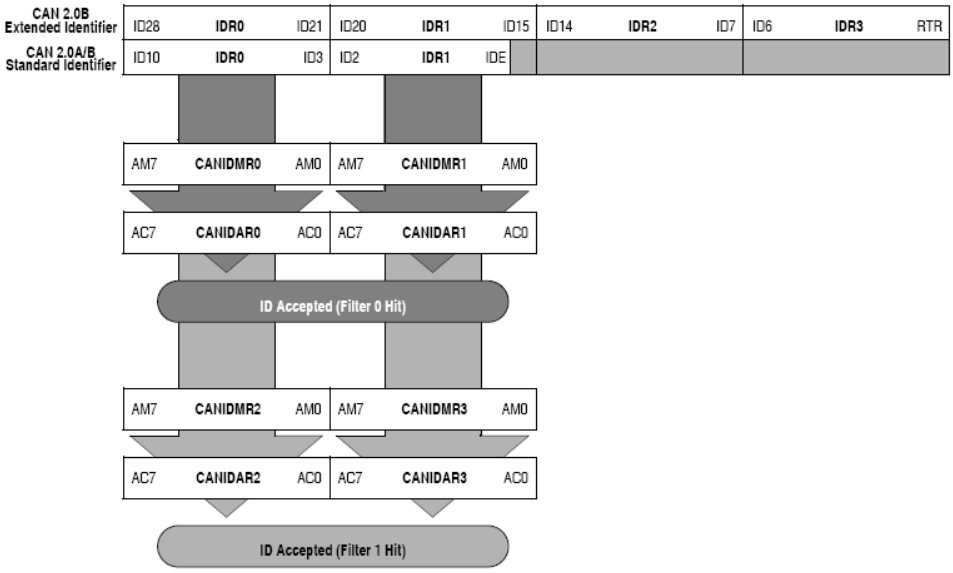
[1] Procesorový modul SHARK DB1 je osazen jedním paměťovým čipem o kapacitě 128 MB. Kromě RAM používá procesor i 64 MB flash paměti.

a potrvzení zrušení. Popis jednotlivých registrů s jejich offsetovými adresami je součástí přílohy B.1.

K nastavení filtru příchozích zpráv se u řadiče MSCAN používá sada šestnácti registrů, osm obsahuje identifikátor(y) zprávy a zbývajících osm obsahuje příslušné masky (viz. příloha B.2). MSCAN umožňuje použít tyto registry čtyřmi různými způsoby:

- jako dva 32-bitové filtry
- jako čtyři 16-bitové filtry
- jako osm 8-bitových filtrů
- nepoužívat filtr vůbec, řadič pak nepřijímá žádné zprávy

Pouze první způsob použití umožňuje filtrování dle celého rozšířeného identifikátoru. Ostatní způsoby využívají k filtrování pouze příslušný počet nejvýznamnějčích bitů identifikátoru. Jednotlivé n-bitové filtry jsou aplikovány postupně, při nalezení shody je zpráva zařazena do přijímacího bufferu, další filtry se již neověřují.



Obrázek 4.2: Schéma filtrování zpráv užitím 16-bitových filtrů

(zdroj: MPC5200B Users Guide, rev 1.0, 2005)

Pokud zpráva projde procesem filtrování a je zařazena do přijímací FIFO fronty, je posunuta do *Foreground Receive Bufferu* - tedy bufferu mapovaného přímo do paměti - a je vygenerováno přerušení (pokud není maskováno). Po načtení zprávy z bufferu je třeba dát řadiči signál, že buffer je volný, vyresetováním příznaku *RXF - Receive Buffer Full.*

Při procesu odesílání je třeba pomocí registru *TBSEL - Transmitt Buffer Selection* zvolit, který buffer přístupný pomocí *TXFG - Foreground Transmit Buffer* (viz. B.3) a po naplnění bufferu dár signál k odeslání zprávy vynulováním odpovídajícícho příznaku *TXE - Transmit Buffer Empty.*

4.2 Průmyslový počítač PC104

Tento průmyslový počítač byl využit při vývoji ovladače pro testování komunikace po CAN sběrnici. PC 104 je modulárně řešený systém, kde se jednotlivé desky (moduly) připojují na sebe a jsou propojeny ISA sběrnicí. Hlavní modul obsahuje procesor Elan520 (kompatibilní s x86 architekturou) operující na frekvenci 133 MHz s vyrovnávací pamětí o velikosti 128 MB, konektor pro CompactFlash kartu, síťový adaptér 10/100 Mbps, a dalšími periferiemi. Na tomto počítači běží OS Linux (jádro 2.6.20 s podporou *Real-Time*), který bootuje z Compact Flash modulu. Do systému se po startu zavádí modul ovladače LinCAN, který obsluhuje čip SJA1000 komunikující po CAN sběrici. Jelikož procesor běží na, na dnešní dobu, nízké frekvenci, nebylo možné použít PC 104 pro testování propustnosti vyvíjeného ovladače pro MSCAN. Průmyslový počítač tak posloužil alespoň pro ověření konzistence přenášených dat (s využitím utilit ovladače LinCAN: *readburst* a *sendburst.*

Obrázek 4.3: Ilustrační foto průmyslového počítače PC104

(zdroj: http://dce.felk.cvut.cz/)

4.2.1 Řadič CAN SJA1000

Průmyslový počítač PC104 je vybaven modulem pro připojení ke CAN sběrnici prostřednictvím CAN řadiče *Philips SJA1000*. Tento diskrétní řadič je nástupcem PCA82C200, proto má stejné pořadí vývodů a v základním režimu (*BasicCAN*) má i stejné chování. V základním režimu řadič podporuje pouze standardní formát zpráv a kvůli kompatibilitě vnějšího interface emuluje všechny registry čipu PCA82C200. Pro použití dle standardní definice [2] je třeba řadič provozovat v *PeliCAN* režimu. Přepnutí je možné provést softwarově nastavením flagu v *clock divider* registru. Pro odesílání používá řadič SJA1000 jeden výstupní buffer o velikosti 13 bytů (pouze pro jednu CAN zprávu). Pro příjem slouží RX FIFO buffer o velikosti 64 bytů, ze kterých je pouze 13 bytů přístupných pro čtení, zbytek využívá pouze *Bit Stream Processor*.

4.3 CAN Analyzer KVASER (PCMCIA)

CAN Analyzer je přídavná PCMCIA karta do notebooku dodávaná společností KVASER. Na kartě jsou dva Philips SJA1000 řadiče pro připojení ke CAN sběrnici. Karta podporuje Real-Time komunikaci po CAN sběrnici. Dodávaný software (*CAN King*) umožňuje jak monitoring sběrnice, tak i aktivní zapojení do komunikace. Karta byla použita jako referenční uzel na CAN sběrnici pro srovnávací měření ovladačů LinCAN a SocketCAN.

Obrázek 4.4: Ilustrační foto PCMCIA karty CAN Analyzeru KVASER

(zdroj: http://www.kvaser.com)

Kapitola 5

Vývoj ovladače LinCAN pro řadič MSCAN

5.1 Příprava HW

Vývoj a testování ovladače pro jádro systému Linux [9] probíhalo na procesorovém modulu s MPC5200 (viz. 4.1) s vstupně-výstupní deskou vyvinutou společností Mikroklima s.r.o, divize MIDAM. Jelikož originální *MIDAM* deska nemá osazené obvody pro fyzické připojení ke CAN sběrnici[1], bylo třeba tyto součástky doosadit. Procesor MPC5200 má vestavěné dva řadiče CAN sběrnice, avšak na *MIDAM* desce je vyveden pouze jeden konektor pro připojení CAN sběrnice. Proto pro pozdější fáze testování byla použita deska vstupů a výstupů vyvinutá na Katedře řídicí techniky nesoucí označení *RYU - EduBoard*, která má vyvedené oba dva řadiče CANu.

[1] Jednalo se o konektor (MSTBV3), budič CAN sběrnice (PCA82C250), galvanický oddělovač (ADUM1201ARZ) a rezistorovo-kondenzátorovou síť

5.2 Příprava vývojového a testovacího prostředí

5.2.1 Kompilace pro cílovou architekturu PowerPC

Pro kompilaci bináních programů na procesor MPC5200 je třeba používat *Cross Compiller* pro PowerPC archtekturu. Architektura PowerPC je poměrně rošířená, a tak je možné sehnat přímo binární balíčky. Pro Cross-kompilaci byly použity balíčky z repozitáře distribuce *Debian* na stránkách Katedry řídicí techiky *(rtime.felk.cvut.cz/debian/)*. Balíčky obsahují nástroje jak pro kompilaci do formátu *ELF* tak i pro kompilaci pro platformu Linux.

5.2.2 Cross-kompilace jádra

Pro vývoj ovladače LinCAN bylo použito jádro vycházející z *Vanilla kernel* verze 2.6.30, které bylo upraveno pro použití na procesoru MPC5200 s vstupně výstupní deskou *MIDAM*. Pro testovací účely bylo použito jádro 2.6.30 s patchem pro MPC5200 a jádro vycházející taktéž z *Vanilla kernelu* avšak verze 2.6.29. Jádro 2.6.29 bylo taktéž upraveno pro použití s procesorem MPC5200 a zároveň opatřeno patchem dodávajícím *RealTime* vlastnosti [10]. Zdroje obou verzí jádra s úpravami pro MPC5200 jsou dostupné v *GIT repozitáři* Katedry řídicí techniky (*git://rtime.felk.cvut.cz/shark/linux.git*) jako příslušné branche. Před vlastní kompilací bylo třeba obě jádra nakonfigurovat pro použití správné verze *Cross-Compileru* a oproti defaultní konfiguraci bylo třeba přidat podporu SocketCANu pro řadič MSCAN.

Pro popis mapování jednotlivých periferií do paměti slouží *Device Tree*, který má textový formát a obsahuje kromě vzájemné závisloti jednotlivých uzlů zařízení a sběrnic v systému i vlastnosti jednotlivých uzlů, jako například adresu, číslo přerušení, jména kompatibilních modulů apod. Při bootu systému je jádru předán binární *Flattened Device Tree*, který obsahuje informace o *Device Tree* ve formátu vhodnějším pro zpracování jádrem. Definice *Device Trees* pro použité desky vstupů a výstupů jsou součástí Git repozitáře se zdroji jader. Tyto definice je třeba před předáním jádru při bootu skompilovat do binárního *FDT* formátu.

5.2.3 Spuštění operačního systému pro MPC5200

Po připojení napájení je možné sledovat inicializaci systému na sériovém *PSC 0* portu vyvedeného na desce vstupů a výstupů. Po provedení *Power On Self Test* naběhne rozhraní u-Boot, kterému je možné předat parametry přes sériovou linku pro boot systému linux. Procesorový modul SHARK je vybaven 64 MB flash paměti, ve které je uloženo jádro a root file system. U-Boot umožňuje zavést systém jak z flash paměti, tak i ze sítě pomocí protokolů TFTP a NFS. Druhý způsob je výhodnější pro vývoj, neboť není nutné při každé změně aktualizovat soubory ve flash paměti, což je relativně pomalé a zároveň zbytečně modifikuje paměť, která má omezený počet přepisů. Pro účely vývoje jsou *u-Bootu* předány parametry pro stažení jádra systému a popisu *Flattened Device Structure* přes TFTP ze serveru a připojení root file systému přes NFS.

5.3 Popis implementace ovladače LinCAN

Implementace nového hardware do LinCANu spočívá v přidání nové desky a případně nového čipu. Čip je, z pohledu LinCANu, vlastní řadič CAN sběrnice. Deska pak představuje větší celek, velmi často se jedná o desku s mikrokontrolérem, který zastřešuje jeden či více čipů. V rámci této práce vznikly dvě desky (podpora originálního daughterboardu vyvinutého spolčností *Mikrolima s.r.o, divize MIDAM* a podpora daughterboardu vyvíjeného na katedře nesoucí označení *RYU - EduBoard*). Základem obou těchto desek je procesorový modul SHARK s procesorem MPC5200 (viz. 4.1) obsahující dva řadiče MSCAN. Jelikož LinCAN obsahuje pouze několik implementací čipů, a MSCAN mezi ně napatřil, bylo nutné implementovat i tento čip.

Pro podporu řadiče *MSCAN*, který je součástí procesoru *MPC5200*, vzniklo tedy pět nových souborů:

soubor	stručný obsah
mpc5200.h	hlavičkový soubor implementovaných desek; Soubor obsahuje deklarace funkcí pro podporu implementované desky *mpc5200* a příslušná makra (např. počet čipů na kartě či frekvence pro výpočet Bus Timing)
mscan.h	hlavičkový soubor implementovaného čipu; Soubor obsahuje makra s adresami všech registrů řadiče MSCAN, enumy popisující obsah jednotlivý registrů a strukturu pro přístup k RX a TX bufferům.
mpc5200.c	implementace desky s oběma řadiči (*RYU*); Soubor obsahuje definice podpůrných funkcí pro práci se systémem s procesorem MPC5200 (např. žádost o přidělení IO paměti a přerušení nebo zápis a čtení bloku paměti)
mpc5200_midam.c	implementace desky pouze s jedním řadičem (*originální MIDAM*); Soubor pouze includuje implementaci s oběma řadiči. Vlastní rozlišení počtu řadičů je řešeno na úrovni maker, která se definují při kompilaci (*-DMIDAM=1*).
mscan.c	Hlavní část práce spočívala ve vytvoření a odladění kódu obsaženého v tomto souboru. Jedná se o práci s řadičem MSCAN, jeho konfigurace pomocí přímého přístupu k registrům řadiče, obsluha jeho požadavků o přerušení a v neposlední řadě řízení komunikace po CAN sběrnici (příjem, odesílání zpráv a nastavení komunikační rychlosti)

Tabulka 5.1: Implementace ovladače - obsah zdrojových kódů

Pro přidání podpory řadiče MSCAN do ovladače LinCAN bylo, kromě vlastní implementace jednotlivých podpůrných funkcí, nutné i modifikovat LinCAN framework, jedná se o modifikace v:

soubor	popis změn
src/boardlist.c	V souboru se dle definovaných maker plní seznam podporovaných desek. Zde přibyly dvě definice pro *mpc5200* a *mpc5200_midam*.
src/Makefile.omk	Soubor je includován při volání *make* a obsahuje názvy desek a pravidla, kterými se při kompilaci jednotlivých desek řídí. Do tohoto souboru byla přidána podpora obou desek a pravidla pro jejich kompilaci.

Tabulka 5.2: Implementace ovladače - modifikace LinCAN frameworku

5.4 Vybrané části implementace

5.4.1 Získání systémových prostředků

Jedním z prvních problémů se úkazalo být použití Flattened Device Structure pro alokaci prostředků pro periferie (viz. 4.1.1). Driver LinCAN (viz. 3.1) pracuje nyní s pevnými adresami zařízení[1], které se modulu předávají při nahrávání do systému. Při použití Flattened Device Structure jsou prostředky přidělovány dynamicky. Během alokace je tedy nutné projít strom zařízení s využitím *Open Firmware* frameworku obsaženém v jádře a nalézt příslušné IO adresy a přerušení obou řadičů (soubor *mpc5200.c*). Jelikož adresa jednotlivých zařízení je uložena ve struktuře *canchip_t*, která v okamžiku dealokace paměti je již uvolněna, je nutné uchovávat přidělené adresy v pomocném poli *chips_addr* (soubor *mpc5200.c*).

[1] Typické pro sběrnici ISA, dnes je již pevná adresace vytlačována technologií Plug and Play.

5.4.2 Obsluha komunikace po sběrnici

5.4.2.1 Nastavení komunikační rychlosti

Řadič MSCAN má pro nastavení *bit timingu* vyhrazené dva registry, které obsahují proměnné *Time Segment 1 a 2* a *Baud Rate Prescaler* - předděličku. Na základě požadavku na konkrétní *baud rate* je vybrána nejlepší kombinace předděličky a obou *Time Segmentů* tak, aby byl minimalizován rozdíl nastavené a požadované rychlosti. Poměr hodnoty *Time Segment 1* ku *Time Segment 2* je dán požadovanou hodnotou na *Sampling Time*, který udává relativní zpoždění bodu, kdy je odečteno napětí, od počátku přijímaného bitu. Použité hodnoty *Sampling Time* pro jednotlivé komunikační rychlosti odpovídají hodnotám doporučeným ve specifikaci CANu [2].

5.4.2.2 Nastavení filtru přijímaných zpráv

Jak je uvedeno v popisu řadiče MSCAN (viz. 4.1.1), obsahuje řadič 16 registrů pro nastavení filtru příchozích zpráv pracující ve čtyřech různých režimech. Pouze jeden režim však umožňuje filtrování zpráv s použitím plné délky rozšířeného identifikátoru. Právě tento režim byl využit při implementaci filtru příchozích zpráv kvůli odlehčení Softwarové části driveru[1]. Jádro ovladače LinCAN při volání *filtch_rq* předává jako parametr výsledek sjednocení filtrů z front uživatelských aplikací, tato kombinace identifikátoru, masky a příznaků zprávy (*MSG_EXT - použití rozšířeného identifikátoru, MSG_RTR - Remote Transfer Request*) je uložena na příslušné pozice obou filtrů.

5.4.2.3 Příjem zpráv

Buffery pro přijímané a odesílané zprávy mají stejný formát, takže bylo možné pro čtení resp. zápis použít strukturu *struct mscan_msg_buffer* obsahující 8-bitové proměnné s identifikátorem zprávy a daty. Po úspěšném přijetí zprávy a jejím umístění do přijímacího FIFO bufferu je vygenerováno přerušení. V obsluze přerušení je načten obsah zprávy z bufferu a vyresetován příznak *RXF* signalizující platnou zprávu v přístupné pozici přijímacího FIFO bufferu, čímž je umožněň příjem další zprávy.

5.4.2.4 Odesílání zpráv

Pro odesílání jsou v řadiči MSCAN připraveny tři buffery, avšak přímý zápis či čtení je umožněno pouze do jednoho z nich určeného nastavením registru *TBSEL*.

[1] Jedná se o režim dvou 32-bitových filtrů, tedy 32-bitových dvojic identifikátor-maska.

Požadavek na odeslání zprávy je generován buď při obsluze volání *write* uživatelskou aplikací, nebo jako reakce na přerušení *TXE - Transmit buffer empty* při odeslání zprávy. Při obsluze požadavku na odeslání zprávy *WakeUp TX* je nejprve zkontrolováno, zdali FIFO fronta obsahuje zprávu připravenou k odeslání, poté je nalezen volný odesílací buffer. V případě, kdy žádný buffer není volný, se ukončí obsluha žádosti a vyčká na její opětovné zavolání během obsluhy přerušení *TXE*. Pokud je nalezen volný TX buffer je obsah odesílané zprávy zkopírován do bufferu, povoleno přerušení při uvolnění použitého bufferu a vyresetován příznak *TXE*, čímž je vydán pokyn k odeslání zprávy. Přerušení je opět zakázáno při obsluze *TXE*. Tato zdánlivě zbytečná práce se zapínáním a vypínáním přerušení pro jednotlivé buffery je důsledkem poněkud nešťastného návrhu řadiče, kdy pro pokyn k odeslání zprávy z bufferu a příznak prázdného bufferu (a tedy žádost o přerušení) je použit jeden a týž příznak *TXE*. Řadič, pokud je příznak *TXE* vynulován, se snaží o odeslání zprávy z bufferu, v případě, kdy je příznak *TXE* nastaven, vyvolává řadič cyklicky přerušení, pokud není maskováno.

5.4.2.5 Obsluha a maskování přerušení

Jak bylo zmíněno v předchozích odstavcíh, obsluhuje ovladač přerušení vyvolané vyprázdněním odesílacího bufferu a příjmem zprávy (naplnění konce přijímací FIFO fronty). Kromě těchto zdrojů přerušení ještě driver reaguje na změnu *Bus state*, tedy přechod např. do stavu *Error Passive* či *Bus Off*, pro oba směry komunikace (*RX i TX*). Změnu stavu sběrnice hlásí ovladač pomocí volání funkce *printk* - chybový výpis jádra[1].

[1] Zpráva je vypsána do systémové konzole (v případě SHARKu je to sériová konzole na PSC0) a sdělení je i přidáno do systémového hlášení *dmesg*.

Kapitola 6

Srovnání ovladačů SocketCAN a LinCAN

6.1 Vlastnosti obou implementací

Porovnání obou ovladačů z hlediska čistoty kódu, případně kvality použitých algoritmů je velmi obtížné. Hodnocení se opírá o návyky a konvence každého programátora. Architektura obou ovladačů je odlišná, LinCAN používá na míru vytvořenou soustavu FIFO front, SocketCAN oproti tomu využívá obecnou vrstvu socketů v jádře. Zatímco LinCAN je vyvíjen tak, aby splňoval nároky na *RealTime* zpracování, hlavní myšlenkou SocketCANu je vytvořit jednoduše a obecně použitelný ovladač. Z hlediska obtížnosti použití se jeví oba ovladače srovnatelně. Pro komunikaci s hardware se používají systémová volání (např. *ioctl* a *setsockopt*), které využívají struktury definované buď přímo v jádře (*can_frame*) nebo v hlavičkovém souboru (*canmsg_t*. Oba ovladače umožňují použití filtru přijímaných zpráv a podporují plně standard CAN 2.0A/B [2].

6.2 Metody srovnání

Z hlediska užití ovladače jsou zajímavé faktory jako zpoždění a propustnost. Porovnáním zpoždění se zabývá *Timing Analysis of Linux CAN Drivers* [11], proto je tato práce zaměřená na srovnání propustnosti obou ovladačů. Srovnání je provedeno na výsledcích série měření. Při měření byla nastavována teoretická rychlost přenosu

pomocí parametrů *wait* a *burst* předané aplikaci *canburst*. Parametr *wait* určuje prodlevu mezi jednotlivými zápisy do otevřeného *file deskriptoru*. Parametr *burst* specifikuje počet struktur předaných při volání *write*.

6.2.1 Testovací aplikace Canburst

Součástí kódů ovladače LinCAN jsou i utility pro měření propustnosti *sendburst* a *readburst*. SocketCAN žádné podobné utility neobsahuje, proto vznikla pro účely této práce aplikace *canburst*. V první verzi pouze odesílala či přijímala zprávy pomocí socketů. Pro lepší porovnání ovladačů LinCAN a SocketCAN bylo vhodné, aby s oběma ovladači pracoval stejný kus kódu. Proto byla aplikace *canburst* rozšířena o podporu ovladače LinCAN. Postupně se rozširovala její funkcionalita až do finální formy, která byla použita pro srovnávací testy obou ovladačů.

Aplikace canburst se skládá z jednoho zdrojového kódu, který pomocí preprocesorových maker produkuje čtyři binární aplikace - aplikace pro odesílaní a příjem zpráv pro oba ovladače. Aplikace pro příjem zpráv otevře a inicializuje příslušný *dev nod* (případně *socket*) a v cyklu čte zprávy z *file descriptoru*. Aplikace pro odesílání zpráv v cyklu odesílá balíky zpráv (*burst* režim), kde využívá možnosti předávat systémovému volání *write* ukazatel na pole struktur, čímž se dosáhne opravdu maximální možného přenosu mezi uživatelskou aplikací a jadry obou driverů.

Jak bylo zmíněno, zaměření výsledné aplikace se řídí definovanými makry.

makro	význam v kódu
SEND=1	Struktura aplikace pro odesílání a příjem je téměř totožná, liší se zejména voláním *read* nebo *write* a nastavováním filtrů pro příjem. Pro úsporu kódu se používá toto makro, místo dvou souborů s téměř shodným obsahem.
LINCAN=1	Práce s LinCANem a SocketCANem se liší v použitých datových strukturách pro výměnu informací (zpráv a filtrů). Pomineme-li tento fakt, jedná se o práci s CAN zprávami. Kód pro odesílání a příjem zpráv obsahuje pro operace se zprávami a filtry makra, jejichž obsah je řízen přepínačem *LINCAN*.

Tabulka 6.1: Canburst - Použitá makra pro řízení překladu

Použití aplikace by mělo být intuitivní a proto se na tomto místě omezíme na popis

pouze jednoho parametru aplikace. Při spuštění aplikace s parametrem *–help* vypíše aplikace všechny možné parametry pro její použití.

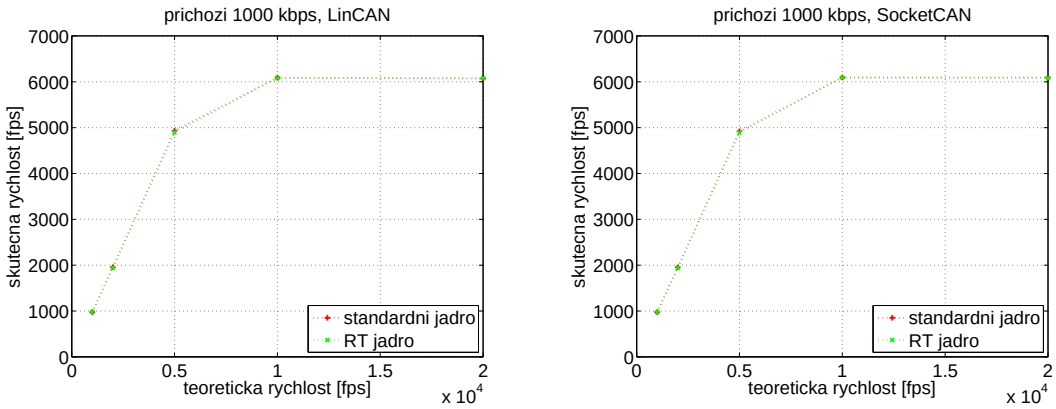
6.3 Varianty testů

6.3.1 Propustnost při různých nastavení bitrate

Toto měření bylo zaměřeno na porovnání chování obou ovladačů v ideálním případě, kdy na sběrnici jsou pouze dva uzly. Všechny vysílané zprávy byly přijímány posluchačem. Nezávisle proměnné v tomto měření představovaly: prodleva mezi balíkem odeslaných zpráv (parametr *wait*), počet zpráv odeslaných najednou (parametr *burst*) a nastavený *baud rate*.

Obrázek 6.1: Blokové schéma zapojení pro měření bitrate

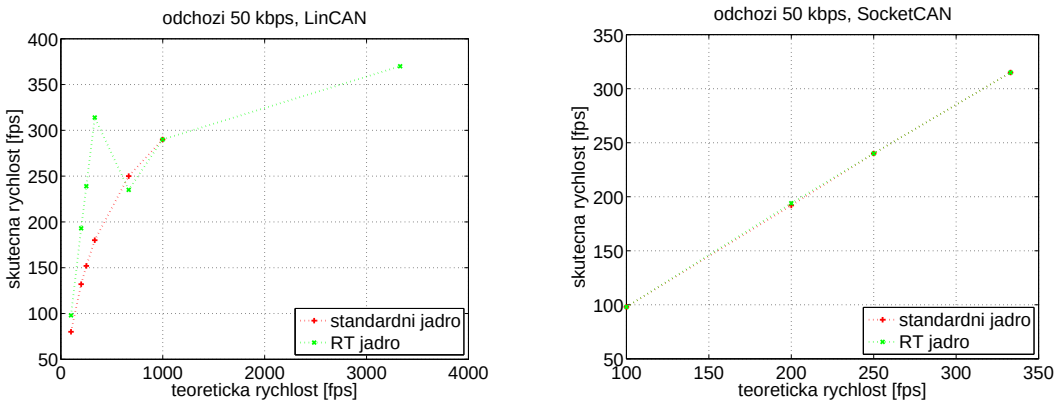
Počet *přijímaných* zpráv za sekundu není ovlivněn použitým jádrem, i oba testované ovladače vykazují stejné výsledky pro patřičné nastavení *baud rate*. Pro ilustraci jsou zde grafy pro měření propustnosti[1] při příjmu pouze pro hodnotu *baud rate* 1000 kbps.



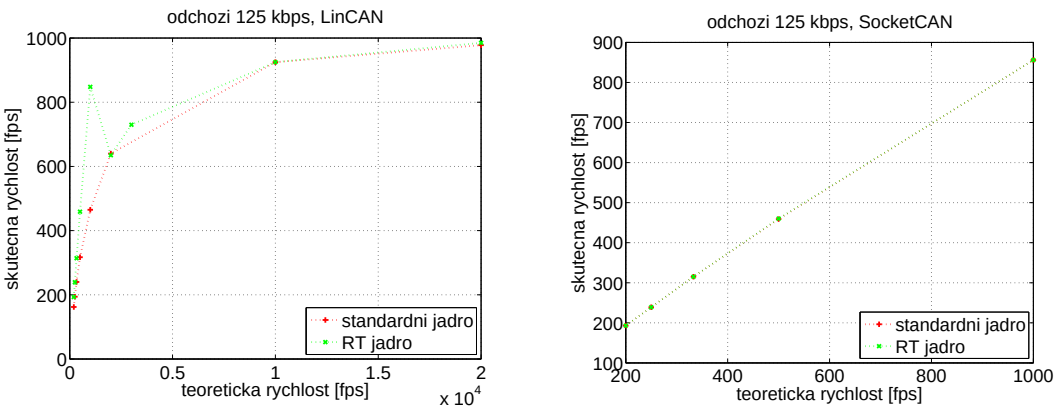
Obrázek 6.2: Závislost teoretické a skutečné propustnosti - příjem, 1000 kbps

[1] fps - *frame per second* - počet přenesených zpráv za sekundu

Při odesílání se použití ruzných jader neprojevilo na charakteristikách ovladače SocketCAN. Rozdíl mezi standardním a RT-jádrem je patrný na grafech ovladače LinCAN. Zejména pro rychlosti 50 a 125 kbps je výrazný rozdíl v propustnosti v místě změny parametru *burst* na hodnotu vyšší než 1 (teoretické rychlosti 333 a 666 fps).

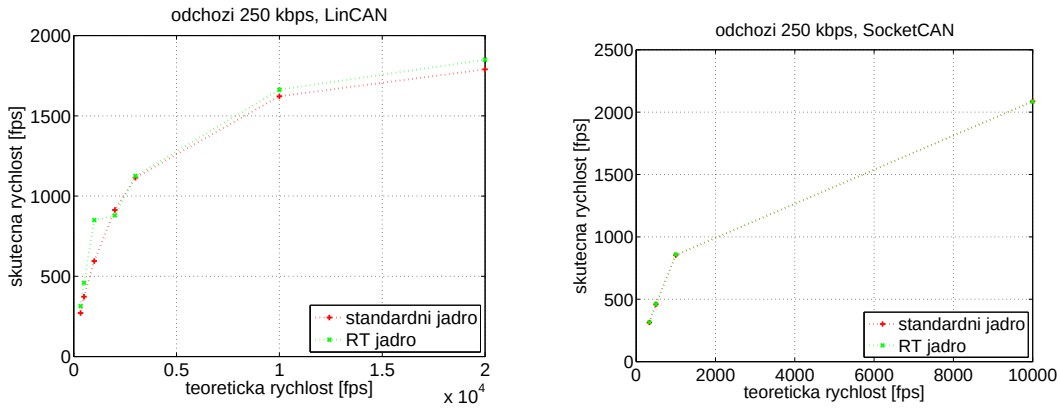


Obrázek 6.3: Závislost teoretické a skutečné propustnosti - odesílání, 50 kbps

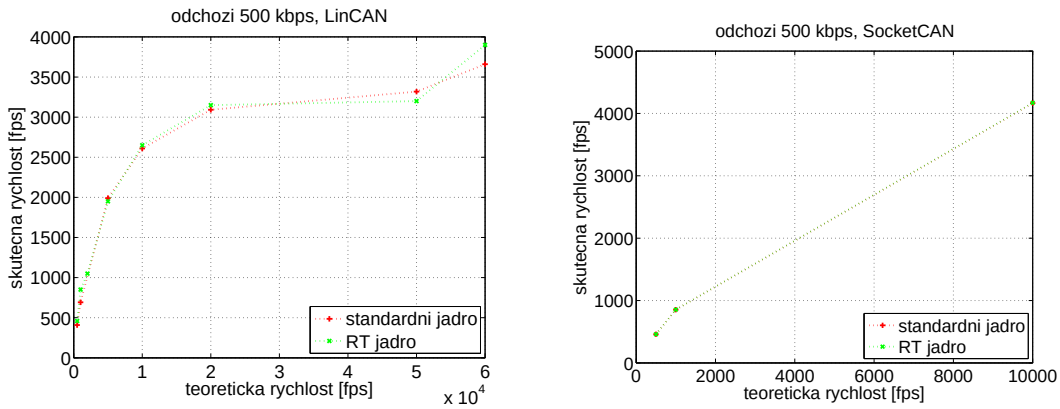


Obrázek 6.4: Závislost teoretické a skutečné propustnosti - odesílání, 125 kbps

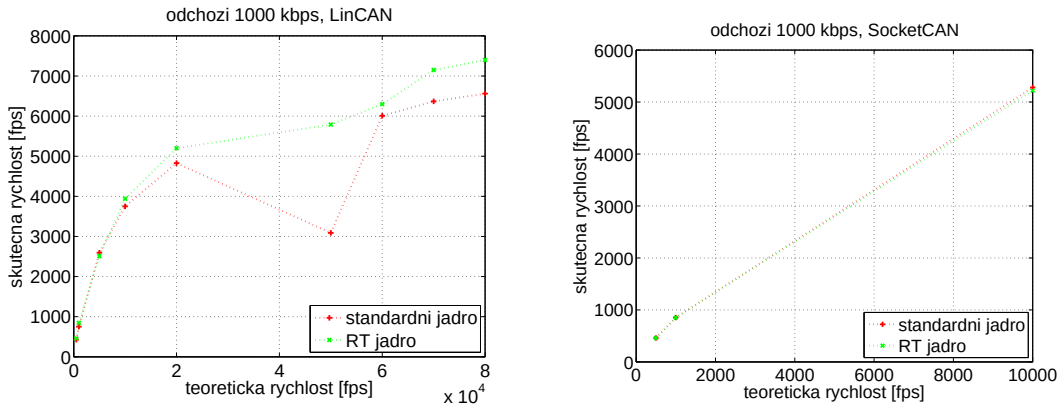
Jádro s aplikovaným Real-Time patchem výrazně zvýšilo propustnost ovladače LinCAN pro přenosy bez využití *burst* režimu.



Obrázek 6.5: Závislost teoretické a skutečné propustnosti - odesílání, 250 kbps



Obrázek 6.6: Závislost teoretické a skutečné propustnosti - odesílání, 500 kbps



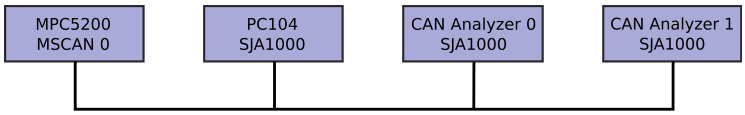
Obrázek 6.7: Závislost teoretické a skutečné propustnosti - odesílání, 1000 kbps

Ovladač SocketCAN dosahuje při zápisu jedné zprávy během volání *write* vyšší maximální propustnosti než LinCAN. Avšak neumožňuje používání *burst* režimu, proti tomu ovladač LinCAN podporuje zápis až 64 zpráv v *burst* režimu, což při použití standardního jádra zvyšuje výrazně jeho propustnost[1]. V absolutním srovnání umožňuje ovladač LinCAN dosáhnout vyšší maximální propustnosti než SocketCAN.

Kompletní seznam hodnot je součástí přílohy (viz. A).

6.3.2 Propustnost zatížené sběrnice

Při tomto měření byly na sběrnici přítomny tři uzly, posluchač, vysílač a zdroj "rušení", který generoval provoz na sběrnici představovaný množstvím zpráv, které posluchač ignoroval. Toto měření mělo sloužit hlavně k porovnání efektivity filtrování zpráv, ta se měla projevit na zátěži systému a následně v počtu přeslechnutých zpráv. Nezávisle proměnnými během tohoto měření byly: rychlost odesílání rušících zpráv (počet zpráv v *burst* režimu a prodleva jejich odesílnání) a rychlost odesílání žádaných zpráv.



Obrázek 6.8: Blokové schéma zapojení pro měření zatížené sběrnice

Při měření nepřekročil počet přenesených zpráv za sekundu hodnotu zhruba 3800 a to ani pro bitrate 1000 kbps. Na sběrnici se objevovaly *Overload frame*, které uměle snižovaly počet přenesených užitečných zpráv. Experimentalné bylo zjištěno, že omezení způsobuje použití obou kanálu CAN Analyzéru. Po odpojení druhého kanálu analyzéru se zvýšila maximální propustnost zpět na maximum naměřené v předchozím bodě. Bohužel bez jednoho kanálu CAN Analyzéru nebylo možné měření provést, neboť druhý kanál sloužil ke generování rušení.

6.3.3 Připojení více aplikací na jeden komunikační objekt

Toto měření porovnává efektivitu rozdělování zpráv dle filtru uživatelské aplikace, které realizováno u obou ovladačů softwarově. Nezávisle proměnné během tohoto

[1] Při použití RT jádra je propustnost s ovladačem LinCAN a použití burst také vyšší, rozdíl ale není tak markantní.

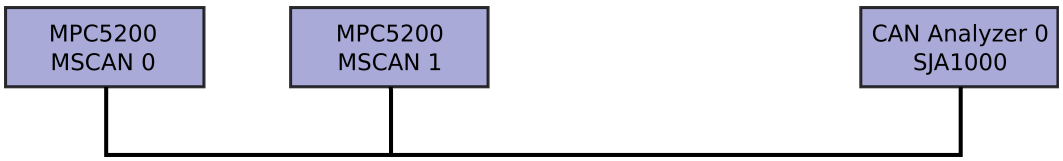
měření jsou: hustota nežádoucích zpráv (zatížení sběrnice), rychlost odesílání zpráv a počet uživatelských aplikací sdílející jeden komunikační objekt.

Obrázek 6.9: Blokové schéma zapojení pro měření při více klientských aplikacích

Pro indikaci zatížení systému byl použit ukazatel *top* s údajem *Load Average*. Během měření však žádný z ovladačů nevykazoval větší zatížení (*Load Average*) než 0.01%, a to ani při použití 10 klientských aplikací. Aby toto měření mělo nějaký význam, bylo by třeba monitorovat systém přesnějším nástrojem. To však je poměrně obtížné, neboť systém na MPC5200 obsahuje pouze nutné nástroje, aby jeho velikost byla co nejmenší.

6.3.4 Propustnost při užití obou řadičů MSCAN

V tomto měření figurují oba dva řadiče MSCAN na procesoru MPC5200. Cílem je odhadnout závislost zatížení systému na počtu ovládaných řadičů pro oba ovladače. Jako nezávislé proměnné byly užity parametry vysílání *wait* a *burst*, pomocí kterých bylo možné přibližně nastavovat počet přenášených zpráv za sekundu.



Obrázek 6.10: Blokové schéma zapojení pro měření při použití obou řadičů MSCAN

Měření byla prováděna pro tři varianty zapojení:

- MSCAN 0 - vysílač, MSCAN 1 - vysílač, CAN Analyzér - posluchač
- MSCAN 0 - vysílač, MSCAN 1 - posluchač, CAN Analyzér - posluchač
- MSCAN 0 - posluchač, MSCAN 1 - posluchač, CAN Analyzér - vysílač

6.3.4.1 Oba řadiče MSCAN ve funkci vysílače

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv
LinCAN	2	1	2 * 425	2 * 30 000
LinCAN	1	1	2 * 724	2 * 50 000
LinCAN	1	2	2 * 1 307	2 * 50 000
LinCAN	1	5	2 * 2 515	2 * 50 000
LinCAN	0	1	2 * 2 210	2 * 50 000
LinCAN	0	5	2 * 3 730	2 * 50 000
SocketCAN	2	1	2 * 455	2 * 30 000
SocketCAN	1	1	2 * 833	2 * 30 000
SocketCAN	0	1	2 * 2 410	2 * 50 000

Tabulka 6.2: Tabulka hodnot - oba řadiče MSCAN ve funkci vysílače, jádro 2.6.30

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv	počet EF[1]
LinCAN	2	1	2 * 458	2 * 30 000	0
LinCAN	1	1	2 * 848	2 * 50 000	0
LinCAN	1	2	2 * 1 250	2 * 50 000	401
LinCAN	1	5	2 * 2 575	2 * 100 000	504
LinCAN	0	1	2 * 3 690	2 * 100 000	0
LinCAN	0	5	2 * 2 750	2 * 100 000	4947
SocketCAN	2	1	2 * 457	2 * 30 000	0
SocketCAN	1	1	2 * 846	2 * 50 000	0
SocketCAN	0	1	2 * 2 900	2 * 50 000	0

Tabulka 6.3: Tabulka hodnot - oba řadiče MSCAN ve funkci vysílače, jádro 2.6.29-rt

Smyslem tohoto měřená bylo zjistit, zdali je propustnost ovlivněna počtem obsluhovaných řadičů ve funkci vysílače. Z hodnot vyplývá, ze pro nižší rychlosti nedochází k žádnému omezení rychlosti odesílání. Pro vyšší teoretické propustnosti je patrný v některých případech pokles skutečné. Ten je způsoben chybami na sběrnici (výskyt *Error Frames*) a naplněním maximální přenosové kapacity sběrnice.

[1] Error Frames

6.3.4.2 Řadiče MSCAN ve funkci vysílače i posluchače

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv
LinCAN	2	1	414	30 000
LinCAN	1	1	706	30 000
LinCAN	1	2	1 262	50 000
LinCAN	1	5	2 430	50 000
LinCAN	0	1	2 428	50 000
LinCAN	0	5	4 820	100 000
SocketCAN	2	1	455	30 000
SocketCAN	1	1	836	30 000
SocketCAN	0	1	2 955	30 000

Tabulka 6.4: Tabulka hodnot - řadiče MSCAN jako vysilač i posluchač, jádro 2.6.30

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv
LinCAN	2	1	458	30 000
LinCAN	1	1	848	30 000
LinCAN	1	2	1 240	50 000
LinCAN	1	5	2 400	50 000
LinCAN	0	1	3 910	75 000
LinCAN	0	5	5 130	75 000
SocketCAN	2	1	459	30 000
SocketCAN	1	1	849	30 000
SocketCAN	0	1	3 235	50 000

Tabulka 6.5: Tabulka hodnot - řadiče MSCAN jako vysilač i posluchač, jádro 2.6.29-rt

Počet přenášených zpráv je nepatrně vyšší u RealTime jádra, avšak vlastní vliv použití obou řadičů na rychlost přenosu není výrazný. Sběrnice není maximálně vytížena a tak nedochází k omezení rychlosti jako v předchozím bodě.

6.3.4.3 Oba řadiče MSCAN ve funkci posluchače

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv
LinCAN	2	1	488	30 000
LinCAN	1	1	972	50 000
LinCAN	1	5	4 895	100 000
LinCAN	0	10	6 080	68 043
SocketCAN	2	1	488	30 000
SocketCAN	1	1	982	50 000
SocketCAN	1	5	4 890	100 000
SocketCAN	1	10	6 090	100 000

Tabulka 6.6: Tabulka hodnot - oba řadiče MSCAN ve funkci posluchače, jádro 2.6.30

ovladač	wait	burst	propustnost [fps]	počet přenesených zpráv
LinCAN	2	1	488	30 000
LinCAN	1	1	972	50 000
LinCAN	1	5	4 890	100 000
LinCAN	0	10	6 090	68 043
SocketCAN	2	1	488	30 000
SocketCAN	1	1	980	50 000
SocketCAN	1	5	4 890	100 000
SocketCAN	1	10	6 090	100 000

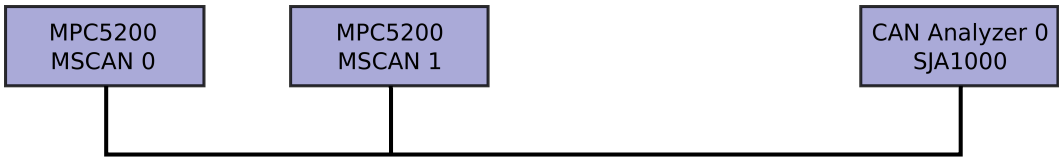
Tabulka 6.7: Tabulka hodnot - oba řadiče MSCAN ve funkci posluchače, jádro 2.6.29-rt

V tomto případě byla srovnávána rychlost příjmu obou řadičů s rychlostí příjmu pouze jednoho řadiče. Srovnáme-li data s rychlostí příjmu jednoho řadiče v 6.3.1, zjistíme, že rychlosti se liší pouze nepatrně. Můžeme tedy konstatovat, že ani LinCAN ani SocketCAN při obsluze více přijímacích zařízení nesnižují maximální počet přenesených zpráv za sekundu.

6.3.5 Spolehlivost při velkém počtu přenášených zpráv

Jednou z hlavních předností sběrnice CAN je její důraz na bezeztrátový přenos zpráv. Toto měření mělo ověřit, zdali oba drivery opravdu "neztrácejí" zprávy. Při testu se přenesl

jeden milion zpráv vysílaných z jednoho řadiče MSCAN. Druhý řadič MSCAN sloužil jako posluchač a jako reference se komunikace účastnil i jeden kanál CAN Analyzéru, také ve funkci posluchače.



Obrázek 6.11: Blokové schéma zapojení pro měření výpadku zpráv

Pro měření byly použity parametry *wait* a *burst* takové, aby ovladače pracovaly s maximální rychlostí přenosu zpráv, kterou umožňují.

ovladač	wait	burst	verze jádra systému	doba přenosu	vypočtená rychlost
SocketCAN	0	1	2.6.30	5:38	2958,58 zpráv/sekundu
LinCAN	0	64	2.6.30	2:44	6097,56 zpráv/sekundu
SocketCAN	0	1	2.6.29-rt	5:26	3067,48 zpráv/sekundu
LinCAN	0	64	2.6.29-rt	2:27	6802,72 zpráv/sekundu

Tabulka 6.8: Tabulka hodnot - test spolehlivosti

Všechny vyslané zprávy byly v pořádku přijaty jak druhým MSCAN řadičem, tak referenčním CAN Analyzérem. Na tomto měření je dobře patrný rozdíl mezi ovladači s ohledem na maximální propustnost. Díky použití *burst* režimu přenesl ovladač LinCAN zprávy za poloviční čas. V tomto testu se i lépe projevil vliv RT patche pro jádro systému.

Kapitola 7

Závěr

Během testů při prorovnávání ovladače LinCAN s ovladačem SocketCAN, byla oveřena funkčnost implementovaného driveru v mnoha směrech. Samotné testy byly zameřeny na typické použití ovladače. Těmto testům zmíněným v práci ještě předcházelo alfa-testování během vývoje, během kterého se také podařilo velké množství neduhů najít a odstranit. V testech proti ovladači SocketCAN testovaný LinCAN poměrně slušně obstál. Celková koncepce obou ovladačů se liší, proto se nedaly očekávat shodné výsledky. V některých částech měl SocketCAN navrch, jinde dominoval LinCAN.

Výsledky je možné shrnout zhruba takto: Při příjmu zpráv se neprojevil rozdíl mezi použitými jádry ani ovladači. Oba drivery dosahovaly srovnatelných výsledků. Použití jádra s RealTime patchem zlepšilo propustnost obou řadiču. Při odesílání bez použití *burst* režimu je rozdíl velmi dobře patrný zejména u ovladače LinCAN. Z hlediska spolehlivosti obstály oba ovladače, při všech testech nedošlo ke ztrátě ani jediné zprávy.

Literatura

[1] Konrad Etschberger, *Controller Area Network*. IXXAT Press, 2001.

[2] Robert Bosch GmbH, *CAN Specification 2.0*, 1991.

[3] Freescale Semiconductor, Inc., *MPC5200B Users Guide, rev 1.0*, 2005.

[4] Ing. Pavel Píša, *Linux/RT-Linux CAN Driver (LinCAN) manual*, 2005.

[5] "SocketCAN Project homepage," http://developer.berlios.de/projects/socketcan/.

[6] "OCERA Project," http://www.ocera.org.

[7] "Linux Kernel," http://www.kernel.org.

[8] "RT CAN Project Documentation," http://www.xenomai.org/documentation/trunk/html/api/group__rtcan.html.

[9] Lukáš Jelínek, *Jádro systému Linux - kompletní průvodce programátora*. Computer Press, a.s., 2008.

[10] "Real-Time Linux Wiki," http://rt.wiki.kernel.org/.

[11] Ing. Michal Sojka, Ing. Pavel Píša, "Timing Analysis of Linux CAN Drivers," 2009. nepublikováno.

Příloha A

Naměřená data - bitrate

A.1 Bitrate 50 kbps

A.1.1 Odesílání

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	10	1	80	5 000
LinCAN	5	1	132	5 000
LinCAN	4	1	152	5 000
LinCAN	3	1	180	5 000
LinCAN	3	2	250	5 000
LinCAN	3	3	290	5 001
SocketCAN	10	1	98	5 000
SocketCAN	5	1	192	5 000
SocketCAN	4	1	240	5 000
SocketCAN	3	1	315	5 000

Tabulka A.1: Tabulka hodnot - odesílání, 50 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	10	1	98	5 000
LinCAN	5	1	193	10 000
LinCAN	4	1	239	10 000
LinCAN	3	1	314	10 000
LinCAN	3	2	235	10 000
LinCAN	3	3	290	10 002
LinCAN	3	10	370	10 000
SocketCAN	10	1	98	5 000
SocketCAN	5	1	194	10 000
SocketCAN	4	1	240	10 000
SocketCAN	3	1	315	10 000

Tabulka A.2: Tabulka hodnot - odesílání, 50 kbps, jádro 2.6.29-rt

A.1.2 Příjem

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	5	1	196	5 000
LinCAN	3	1	328	5 000
LinCAN	3	2	407	3 499
LinCAN	3	3	407	2 459
SocketCAN	5	1	197	5 000
SocketCAN	3	1	327	5 000
SocketCAN	3	2	408	3 488
SocketCAN	3	3	408	2 457

Tabulka A.3: Tabulka hodnot - příjem, 50 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	5	1	196	5 000
LinCAN	3	1	326	5 000
LinCAN	3	2	407	3 510
LinCAN	3	3	407	2 470
SocketCAN	5	1	196	5 000
SocketCAN	3	1	325	5 000
SocketCAN	3	2	407	3 524
SocketCAN	3	3	408	2 471

Tabulka A.4: Tabulka hodnot - příjem, 50 kbps, jádro 2.6.29-rt

A.2 Bitrate 125 kbps

A.2.1 Odesílání

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	5	1	162	10 000
LinCAN	4	1	194	10 000
LinCAN	3	1	240	10 000
LinCAN	2	1	317	10 000
LinCAN	1	1	465	10 000
LinCAN	1	2	640	10 000
LinCAN	1	10	925	10 000
LinCAN	1	20	978	10 000
SocketCAN	5	1	193	10 000
SocketCAN	4	1	239	10 000
SocketCAN	3	1	315	10 000
SocketCAN	2	1	459	10 000
SocketCAN	1	1	856	10 000

Tabulka A.5: Tabulka hodnot - odesílání, 125 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	5	1	193	10 000
LinCAN	4	1	239	10 000
LinCAN	3	1	314	10 000
LinCAN	2	1	459	10 000
LinCAN	1	1	848	20 000
LinCAN	1	2	635	20 000
LinCAN	1	3	730	20 001
LinCAN	1	10	925	20 000
LinCAN	1	20	985	20 000
SocketCAN	5	1	193	10 000
SocketCAN	4	1	239	10 000
SocketCAN	3	1	315	10 000
SocketCAN	2	1	460	10 000
SocketCAN	1	1	856	20 000

Tabulka A.6: Tabulka hodnot - odesílání, 125 kbps, jádro 2.6.29-rt

A.2.2 Příjem

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	480	5 000
LinCAN	1	1	975	10 000
LinCAN	1	2	994	5 521
LinCAN	1	5	995	2 436
SocketCAN	2	1	490	10 000
SocketCAN	1	1	970	10 000
SocketCAN	1	2	995	5 506
SocketCAN	1	5	995	2 476

Tabulka A.7: Tabulka hodnot - příjem, 125 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	486	5 000
LinCAN	1	1	972	10 000
LinCAN	1	2	988	5 577
LinCAN	1	5	900	2 437
SocketCAN	2	1	487	10 000
SocketCAN	1	1	970	10 000
SocketCAN	1	2	995	5 512
SocketCAN	1	5	995	2 439

Tabulka A.8: Tabulka hodnot - příjem, 125 kbps, jádro 2.6.29-rt

A.3 Bitrate 250 kbps

A.3.1 Odesílání

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	3	1	271	10 000
LinCAN	2	1	373	10 000
LinCAN	1	1	595	10 000
LinCAN	1	2	913	10 000
LinCAN	1	3	1 115	20 001
LinCAN	1	10	1 622	20 000
LinCAN	1	20	1 790	20 000
SocketCAN	3	1	314	10 000
SocketCAN	2	1	460	10 000
SocketCAN	1	1	853	10 000
SocketCAN	0	1	2 087	10 000

Tabulka A.9: Tabulka hodnot - odesílání, 250 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	3	1	314	20 000
LinCAN	2	1	459	20 000
LinCAN	1	1	850	20 000
LinCAN	1	2	880	20 000
LinCAN	1	3	1 125	20 001
LinCAN	1	10	1 663	30 000
LinCAN	1	20	1 850	30 000
SocketCAN	3	1	315	20 000
SocketCAN	2	1	461	20 000
SocketCAN	1	1	858	20 000
SocketCAN	0	1	2 084	20 000

Tabulka A.10: Tabulka hodnot - odesílání, 250 kbps, jádro 2.6.29-rt

A.3.2 Příjem

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	980	10 000
LinCAN	1	2	1 917	20 000
LinCAN	1	3	1 920	13 691
LinCAN	1	5	1 920	8 431
SocketCAN	1	1	978	20 000
SocketCAN	1	2	1 910	20 000
SocketCAN	1	3	1 920	13 744
SocketCAN	1	5	1 920	8 397

Tabulka A.11: Tabulka hodnot - příjem, 250 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	971	10 000
LinCAN	1	2	1 910	20 000
LinCAN	1	3	1 910	13 859
LinCAN	1	5	1 910	8 472
SocketCAN	1	1	978	20 000
SocketCAN	1	2	1 910	20 000
SocketCAN	1	3	1 920	13 828
SocketCAN	1	5	1 920	8 547

Tabulka A.12: Tabulka hodnot - příjem, 250 kbps, jádro 2.6.29-rt

A.4 Bitrate 500 kbps

A.4.1 Odesílání

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	409	20 000
LinCAN	1	1	692	20 000
LinCAN	1	5	1 990	20 000
LinCAN	1	10	2 610	30 000
LinCAN	1	20	3 092	30 000
LinCAN	0	5	3 318	40 000
LinCAN	0	20	3 660	40 000
SocketCAN	2	1	459	30 000
SocketCAN	1	1	853	30 000
SocketCAN	0	1	4 165	30 000

Tabulka A.13: Tabulka hodnot - odesílání, 500 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	459	30 000
LinCAN	1	1	850	30 000
LinCAN	1	2	1 050	30 000
LinCAN	1	5	1 950	30 000
LinCAN	1	10	2 650	40 000
LinCAN	1	20	3 150	40 000
LinCAN	0	5	3 200	40 000
LinCAN	0	20	3 900	40 000
SocketCAN	2	1	460	30 000
SocketCAN	1	1	855	30 000
SocketCAN	0	1	4 170	30 000

Tabulka A.14: Tabulka hodnot - odesílání, 500 kbps, jádro 2.6.29-rt

A.4.2 Příjem

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	976	20 000
LinCAN	1	2	1 980	20 000
LinCAN	1	5	3 530	15 673
LinCAN	1	10	3 530	8 080
SocketCAN	1	1	980	20 000
SocketCAN	1	2	1 970	20 000
SocketCAN	1	5	3 530	15 580
SocketCAN	1	10	3 530	7 888

Tabulka A.15: Tabulka hodnot - příjem, 500 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	972	20 000
LinCAN	1	2	1 950	20 000
LinCAN	1	5	3 530	15 358
LinCAN	1	10	3 540	7 935
SocketCAN	1	1	970	20 000
SocketCAN	1	2	1 960	20 000
SocketCAN	1	5	3 530	15 333
SocketCAN	1	10	3 540	8 021

Tabulka A.16: Tabulka hodnot - příjem, 500 kbps, jádro 2.6.29-rt

A.5 Bitrate 1000 kbps

A.5.1 Odesílání

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	428	20 000
LinCAN	1	1	752	20 000
LinCAN	1	5	2 595	40 000
LinCAN	1	10	3 750	40 000
LinCAN	1	20	4 828	75 000
LinCAN	0	1	3 090	75 000
LinCAN	0	10	6 010	75 000
LinCAN	0	20	6 368	75 000
LinCAN	0	40	6 560	75 000
SocketCAN	2	1	460	20 000
SocketCAN	1	1	853	30 000
SocketCAN	0	1	5 280	75 000

Tabulka A.17: Tabulka hodnot - odesílání, 1000 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	2	1	459	20 000
LinCAN	1	1	849	20 000
LinCAN	1	5	2 510	30 000
LinCAN	1	10	3 940	40 000
LinCAN	1	20	5 200	50 000
LinCAN	0	1	5 790	50 000
LinCAN	0	10	6 300	50 000
LinCAN	0	20	7 150	75 000
LinCAN	0	40	7 400	75 000
SocketCAN	2	1	459	20 000
SocketCAN	1	1	851	20 000
SocketCAN	0	1	5 220	50 000

Tabulka A.18: Tabulka hodnot - odesílání, 1000 kbps, jádro 2.6.29-rt

A.5.2 Příjem

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	975	20 000
LinCAN	1	2	1 955	20 000
LinCAN	1	5	4 930	50 000
LinCAN	1	10	6 090	33 858
LinCAN	1	20	6 070	18 676
SocketCAN	1	1	975	20 000
SocketCAN	1	2	1 953	20 000
SocketCAN	1	5	4 920	50 000
SocketCAN	1	10	6 094	34 054
SocketCAN	1	20	6 093	20 114

Tabulka A.19: Tabulka hodnot - příjem, 1000 kbps, jádro 2.6.30

ovladač	wait	burst	přenosová rychlost [fps]	počet přenesených zpráv
LinCAN	1	1	975	20 000
LinCAN	1	2	1 940	20 000
LinCAN	1	5	4 890	50 000
LinCAN	1	10	6 080	33 982
LinCAN	1	20	6 080	19 386
SocketCAN	1	1	972	20 000
SocketCAN	1	2	1 940	20 000
SocketCAN	1	5	4 890	50 000
SocketCAN	1	10	6 090	33 713
SocketCAN	1	20	6 080	20 445

Tabulka A.20: Tabulka hodnot - příjem, 1000 kbps, jádro 2.6.29-rt

Příloha B

Mapování registrů řadiče MSCAN

paměťový offset	název	popis
0x00	**CTL0**	**Control Register 0**
0x01	**CTL1**	**Control Register 1**
0x04	**BTR0**	**Bus Timing Register 0**
0x05	**BTR1**	**Bus Timing Register 1**
0x08	**RFLG**	**Receiver Flag Register**
0x09	**RIER**	**Receiver Interrupt Enable Register**
0x0C	**TFLG**	**Transmitter Flag Register**
0x0D	**TIER**	**Transmitter Interrupt Enable Register**
0x10	**TARQ**	**Transmitter Abort Request Register**
0x11	**TAAK**	**Transmitter Abort Acknowledge Register**
0x14	**TBSEL**	**Transmit Buffer Selection**
0x15	**IDAC**	**Identifier Acceptance Control Register**
0x1C	**RXERR**	**Receive Error Counter Register**
0x1D	**TXERR**	**Transmit Error Counter Register**

Tabulka B.1: Mapování základních registrů řadiče MSCAN do paměti

paměťový offset	název	popis
0x20	**IDAR0**	**Identifier Acceptance Register 0**
0x21	**IDAR1**	**Identifier Acceptance Register 1**
0x24	**IDAR2**	**Identifier Acceptance Register 2**
0x25	**IDAR3**	**Identifier Acceptance Register 3**
0x28	**IDAM0**	**Identifier Mask Register 0**
0x29	**IDAM1**	**Identifier Mask Register 1**
0x2C	**IDAM2**	**Identifier Mask Register 2**
0x2D	**IDAM3**	**Identifier Mask Register 3**
0x30	**IDAR4**	**Identifier Acceptance Register 4**
0x31	**IDAR5**	**Identifier Acceptance Register 5**
0x34	**IDAR6**	**Identifier Acceptance Register 6**
0x35	**IDAR7**	**Identifier Acceptance Register 7**
0x38	**IDAM4**	**Identifier Mask Register 4**
0x39	**IDAM5**	**Identifier Mask Register 5**
0x3C	**IDAM6**	**Identifier Mask Register 6**
0x3D	**IDAM7**	**Identifier Mask Register 7**

Tabulka B.2: Mapování registrů filtru příchozích zpráv řadiče MSCAN do paměti

paměťový offset	název	popis
0x40 - 0x5F	**RXFG**	**Foreground Receive Buffer**
0x60 - 0x7F	**TXFG**	**Foreground Transmit Buffer**

Tabulka B.3: Mapování registrů RX a TX bufferů řadiče MSCAN do paměti

Uspořádání bitů v jednotlivých registrech je blíže popsáno v technické specifikaci procesoru MPC5200, která je součástí přiloženého CD [3].

Příloha C

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy, dokumenty použité při návrhu driveru a aplikace použité při vývoji.

./docs:

02_Bosch_Can_specification_20A.pdf	[2]
02_Bosch_Can_specification_20B.pdf	[2]
03_Freescale_Semiconductor_MPC5200B.pdf	[3]
04_Pavel_Pisa_LinCAN_manual.pdf	[4]
11_Pisa_Sojka_Can_timing_analysis.pdf	[11]

./docs/doplnkove:

Low_Level_CAN_Framework-2006-02-20.pdf
SHARK_V1_1_DOC_CZ.pdf

./src/canburst:

canburst.tar.gz

./src/lincan:

lincan_snapshot_2009-dec.tar.gz

./dp_2010_martin_petera.pdf