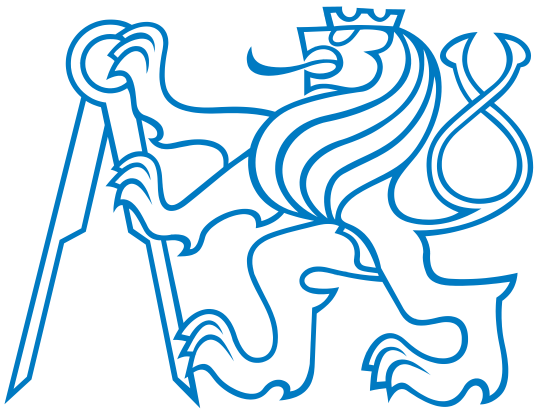
České vysoké učení technické v Praze
Fakulta elektrotechnická



LinCAN driver pro řadič Renesas HCAN2

Bakalářská Práce

Praha, 2007 **Autor: Martin Petera**

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 _____ _____

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 důvěru ve mne vloženou. Speciální poděkování patří také Ing. Pavlovi Píšovi za pomoc s řešením problémů s OS Linux a jádrem driveru LinCAN. V neposlední řadě patří poděkování také mé rodině a spolužákovi Petrovi Svobodovi za morální podporu a nápady na řešení.

Abstrakt

V této bakalářské práci je popsán postup vývoje driveru pro řadič Renesas HCAN2 komunikující po CAN sběrnici. Tento řadič je součástí čipu s RISC procesorem SH7760, který je umístěn na desce s vstupními-výstupními zařízeními EXM32. Tato deska se používá v několika projektech na Katedře Řídící Techniky Fakulty Elektrotechnické, ČVUT Praha. V první části práce naleznete stručný popis CAN sběrnice, vnitřní struktury driveru LinCAN a EXM32 desky s SH7760 procesorovým modulem. V druhé části pak popisuji vlastní implementaci driveru pro řadič HCAN2.

Abstract

This bachelor thesis describes development of a driver for the Renesas HCAN2 controler used for communication over CAN bus. This controler is a part of an embedded system with the RISC processor SH7760, which is used as a main proccessor unit of mother board EXM32. Module EXM32 with SH7760 proccessor board is deployed in projects of Department of Control Engineering, Faculty of Eletrical engineering, Czech Technical University in Prague. The first part of this document describes the CAN bus, inner structure of the LinCAN driver and brings a short description of the EXM32 board with SH7760 proccessor module. The second part is focused on the implementation of the driver itself.

vložit originální zadáníííííííííííííííí !!!!!

Obsah

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) . . . 3

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 . . . 6

2.3.2.1 Zpráva se standardním identifikátorem . . . 7

2.3.2.2 Zpráva s rozšířeným identifikátorem . . . 8

2.3.3 Odeslání zprávy . . . 8

2.3.4 Přijetí zprávy . . . 9

3 Driver LinCAN **10**

3.1 Popis . . . 10

3.2 Fronty zpráv . . . 11

4 Použitý hardware **12**
4.1 Základní deska EXM32 . . . 12
4.2 Procesor SH7760 . . . 13
4.3 Řadič HCAN2 . . . 13
4.3.1 Struktura řadiče . . . 14
4.3.2 Struktura komunikačního objektu - mailboxu . . . 15
4.3.2.1 Maska příchozích zpráv . . . 16
4.4 Ostatní hardware . . . 17
4.4.1 PC104 s řadičem CAN SJA1000 . . . 17
4.4.2 CAN Analyzer - KVASER . . . 17

5 Příprava EXM32-SH7760 s řadičem HCAN2 **18**
5.1 Cross-Kompilace . . . 18
5.2 Nové jádro 2.6.21 . . . 19
5.2.1 Cross-Kompilace a výchozí nastavení . . . 19
5.2.2 Změna boot sektoru - LILO . . . 20
5.2.3 Přenastavení parametrů jádra v lilo.conf . . . 20

6 Implementace driveru LinCAN pro řadič HCAN2 **21**
6.1 Popis . . . 21
6.2 Implementace . . . 22
6.2.1 Změny v config.omk a config.omk-default . . . 22
6.2.2 Změny v src/boardlist.c . . . 22
6.2.3 Obsah include/sh7760.h . . . 23
6.2.4 Obsah include/hcan2.h . . . 23
6.2.5 Obsah src/sh7760.c . . . 23
6.2.6 Obsah src/hcan2.c . . . 24

7 Závěr **29**

Literatura **31**

A Schémata **I**
A.1 Blokové schéma procesoru SH7760 . . . I
A.2 Rozmístění konektorů na základní desce EXM32 MB Lite . . . II

B Popis použití ovladače kontroleru HCAN2 **III**

B.1 Aplikace patche HCAN2 pro ovladač LinCAN III

B.2 Nastavení kompilace ovladače LinCAN IV

B.3 Kompilace ovladače LinCAN a přidání modulu IV

C Obsah přiloženého CD **V**

Kapitola 1

Úvod

Sběrnice CAN (*Controller Area Network*, viz. [3]), definovaná *normou ISO 11898 a CAN specification 2.0B*, 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 Renesas SH7760 (viz. [4]) pro základní desku EXM32. Tento systém je používán v projektech na Katedře řídící techniky Fakulty Elektrotechnické, Českého Vysokého Učení Technického v Praze. Tato práce se zabývá návrhem a následnou implementací ovladače pro operační systémy Linux a Real-Time Linux v rámci projektu LinCAN (viz. [2]). LinCAN je ovladač pro řadiče CAN vyvíjený jako součást projektu OCERA [6].

V první části práce naleznete popis CAN sběrnice a její fyzické vrstvy, stručný úvod k driveru LinCAN a seznámení s hardwarem, který byl použit při tvorbě a verifikaci funkčnosti ovladače. V další části je popsána příprava systému EXM32 s procesorem SH7760, konkrétně aktualizace jádra systému Linux, neboť bylo třeba vyvíjet ovladač na co možná nejaktuálnější verzi. Tato aktualizace se neobešla bez komplikací, které jsou podrobněji zmíněny v kapitole 5. Popis ovladače řadiče HCAN2 je umístěn jako poslední kapitola této práce (kapitola 6).

Kapitola 2

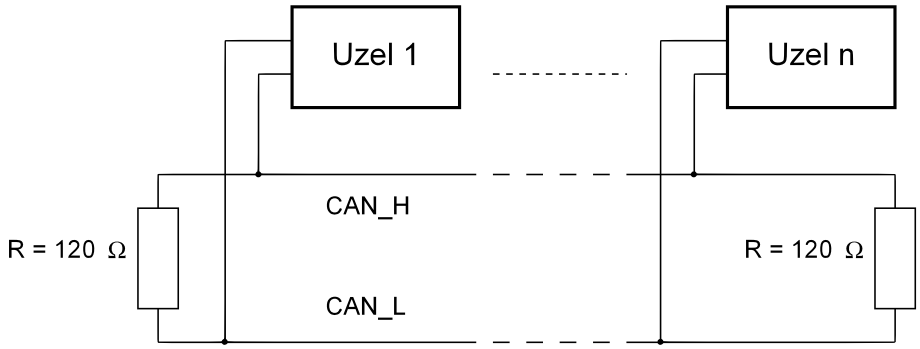
CAN - Controller Area Network

2.1 Popis sběrnice

CAN je průmyslová seriová multimaster sběrnice s rychlostí přenosu až 1 Mb/s při maximální délce 40 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. Pokud uzel posílá velké množství chybných zpráv, dojde k jeho odpojení od sběrnice a tím k zajištění průchodnosti. 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 (PDF je součástí přílohy)[5] a v témže roce vznikl standard ISO 11898, který byl v roce 1995 rozšířen o *Extended Frame* formát.

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 - viz 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 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

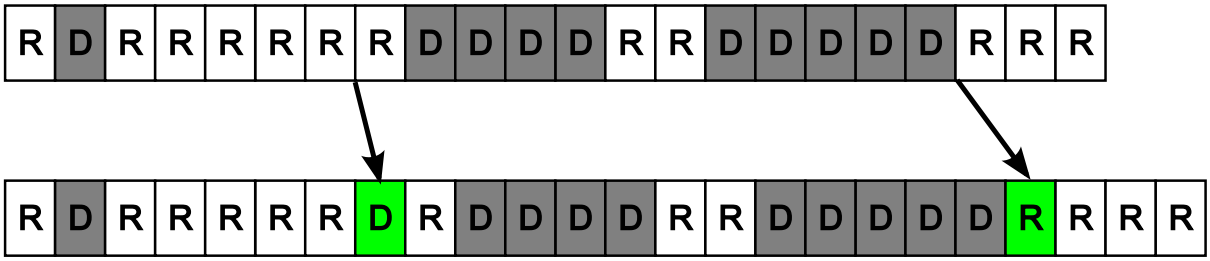
Každý vysílající uzel zároveň sleduje logickou úroveň na sběrnici. Pokud detekuje jinou log. úroveň než vysílá, ukončí vysílání a až do konce přenosu zprávy působí jako posluchač. Tento případ může nastat pouze v případě, že jeden uzel vysílá recesivní úroveň a jiný uzel vysílá dominantní úroveň.

2.2.1.2 Cyclic Redundancy Check (CRC)

Každá zpráva obsahuje 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 $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$.

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 *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.

2.3 Zprávy

Komunikace po sběrnici probíhá pomocí zpráv (rámců). Rozlišují se 4 typy zpráv: *Data Frame (2.3.1.1)*, *Remote Request Frame (2.3.1.2)*, *Error Frame (2.3.1.3)* a *Overload Frame (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.

2.3.1 Typy rámců

2.3.1.1 Data Frame

Rámec 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, a neobsahuje část *Data Field*, i když v části *Control Field* může být počet datových bitů nenulový. Hodnota DLC (*Data Length Code*) udává počet, kolik datových bytů zařízení *master* očekává.

2.3.1.3 Error Frame

Error Frame se skládá ze dvou částí: *Error Flag* a *Error Delimiter*. Error flag je šest po sobě následujících bitů - buď recesivní úrovně, vysílá-li je zařízení v *Error Passive* módu, nebo dominantní úrovně, pokud zařízení, které detekovalo chybu je v *Error Active* módu. Při vysílání *Error Flag* není pravidlo *Bit Stuffing (2.2.1.3)* aplikováno, což detekují ostatní zařízení na sběrnici jako chybu. Po odvysílání prvních šesti bitů *Error Flag* odešle zařízení recesivní úroveň 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 se skládá ze dvou částí zvaných *Overload Flag* a *Overload Delimiter*. Overload flag je šest po sobě následujících bitů dominantní úrovně (stejně jako *Error Active Flag*). 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:

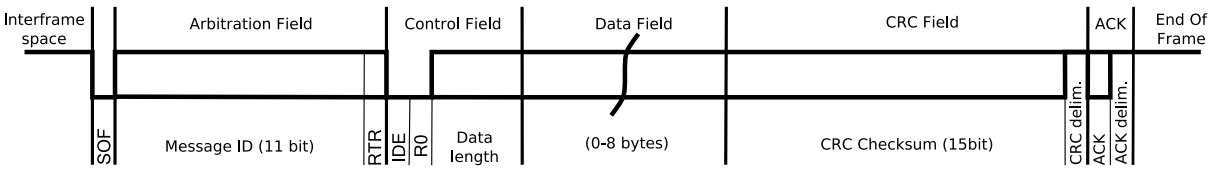
- Uzel, který přijímal zprávu, žádá o krátkou prodlevu před vysíláním dalšího *Data* nebo *Remote Frame*.
- Při detekci dominantních bitů na prvním a druhém místě *Interframe Space* (prostor mezi *Data* a *Remote Frame*).
- Pokud uzel zjistí dominantní úroveň posledního bitu části *Error* nebo *Overload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

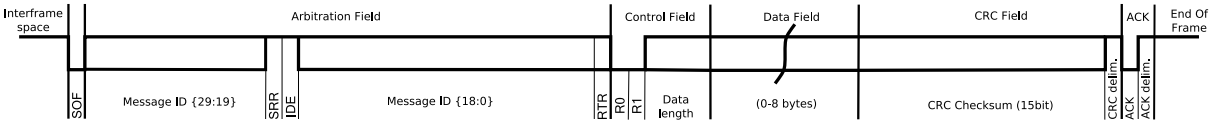
Popis jednotlivých částí zprávy:

- **Start of Frame:** Začátek zprávy je vždy vysílán jako dominantní úroveň. Před vysláním čeká *master* minimálně na tři bity recesivní úrovně, pak zahájí vysílání. Pokud uzel vysílal předchozí *Data Frame* nebo *Remote Frame*, čeká osm bitů před zahájením vysílání. Umožní tak ostatním uzlům zahájit vysílání dříve.
- **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 rozpoznání prvního ACK bitu od *CRC Field*).
- **ACK Field:** Je potvrzení přijetí nepoškozené zprávy a skládá ze ze dvou bitů. *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 pouze informaci zdali *slave* přijal zprávu neporušenou, nevypovídá nic o procesu filtrace zprávy pomocí LAF (*Local Acceptance Filter* - viz. 2.3.4).

- **End of Frame:** Indikuje konec zprávy a je vysílán jako 7 recesivních bitů.

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.

2.3.3 Odeslání zprávy

Data Frame nebo *Remote Request Frame* začínají bitem *Start of Frame* dominantní úrovně, před kterým musí být alespoň tři bity recesivní úrovně. Po *SOF* následuje 11 bitů identifikátoru během kterých probíhá arbitrace přístupu ke sběrnici. Pro arbitraci je použito pravidla *Monitoringu* 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

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 *všech Mailboxů*. Pokud najde shodu, umístí zprávu do *mailboxu* a provede příslušné kroky k oznámení přijaté zprávy (zpravidla vyvolá přerušení).

Kapitola 3

Driver LinCAN

3.1 Popis

LinCAN (viz. [2]) 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. Vnitřně je driver řešen pomocí FIFO front zpráv, které se používají pro odesílání a příjem zpráv. Tuto jeho vlastnost zláště ocení uživatelé, pokud přistupují k jednomu objektu z více aplikací. LinCAN je navžen tak, aby vyhovoval požadavkům na *Real-Time* operační systém, jeho vnitřní metody jsou tedy optimalizovány pro co nejrychlejší obsluhu požadavků.

Jednotlivé funkční bloky driveru lze rozdělit do těchto částí:

- **jádro driveru:** Tato část řeší napojení na systém, přidávání modulu do systému, komunikaci s driverem pomocí systémových volání *open, close, read, write, select, ioctl* z userspace, vnitřní FIFO fronty zpráv a podle požadované operace z *userspace* volá příslušné metody odpovídající použitému hardware. Obecně lze i tento základ rozdělit na pomocné funkce, systém pro propojování a práci s frontami zpráv, obecnou část a část realizující rozhraní pro integraci do operačního systému *Linux* a *RT-Linux*.

- **podpora jednotlivých CAN kontrolerů:** Tato část obsahuje kontrolery se kterými je LinCAN schopen pracovat. Každý kontroler musí implementovat metody pro práci s ním (konfigurace, nastavení rychlosti, příjem a odesílání zpráv, obsluhu přerušení). Tyto metody jsou volány jádrem driveru na základě požadavků, ať z userspace klientskou aplikací - *read, write*, nebo přímo požadavkem systému, například přidáním modulu nebo požadavkem na přerušení.

- **podpora jednotlivých karet:** Tato část tvoří nejpočetnější skupinu a obsahuje metody specifické pro hardware, na kterém je umístěn kontroler umožňující komunikaci po CAN sběrnici. Každá karta musí implementovat základní metody, zejména pro incializaci zařízení, které jsou volány jádrem driveru. Po nastavení výchozích hodnot je již většina volání jádra driveru směřována na konkrétní metody kontroleru.

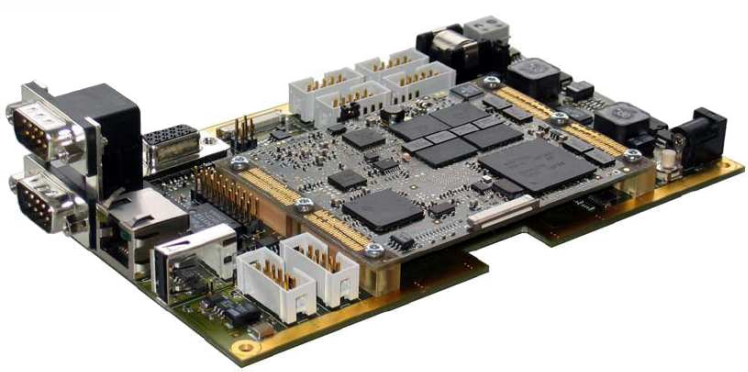
3.2 Fronty zpráv

Fronty slouží k výměně zpráv mezi uživatelskou aplikací (přes zařízení v systému */dev/canX*) a obslužnými rutinami kontroleru. Obsluha kontroleru po přijetí zprávy přidá tuto zprávu do fronty voláním *canque_filter_msg2edges*. Při odesílání zprávy se nejdříve obsluha čipu přesvědčí zavoláním metody *canque_test_outslot*, zda-li je ve frontě zpráva k odeslání. Při chybě, nebo pokud fronta neobsahuje žádnou zprávu k odeslání, vrací metoda záporné číslo. Zpráva čekající na odeslání je přístupna přes strukturu *obj->tx_slot->msg*.

Kapitola 4

Použitý hardware

4.1 Základní deska EXM32



Obrázek 4.1: Deska EXM32 s modulem SH7760 (převzato z [4])

Mateřská deska EXM32 je navržena tak, aby odolávala vnějším rušivým vlivům a byla schopna pracovat v průmyslovém (často velmi zarušeném) prostředí. Na desce EXM32 je umístěn univerzální konektor pro připojení procesorového a dalších rozšiřujících modulů a vstupní/výstupní porty například CAN nebo USB. Rošiřujícími moduly mohou být například WiFi síťový adaptér nebo gigabitový přenosový interface. Na použité desce byl přidán pouze modul s procesorem SH7760 (viz. 4.2). Kromě konektorů CAN sběrnice (*2 x IDC - 10 pinů*) jsou na základní desce i konektory RS232 (*2 x DSUB - 9 pinů a 4 x IDC - 10 pinů*), USB (*USB Typ A*), síťového rozhraní (*RJ45*), VGA (*DSUB - 15 pinů*), LCD (*FFC/FPC - 31 pinů, TFT LVTTL*) a zásuvka pro *Compact Flash* paměťovou kartu.

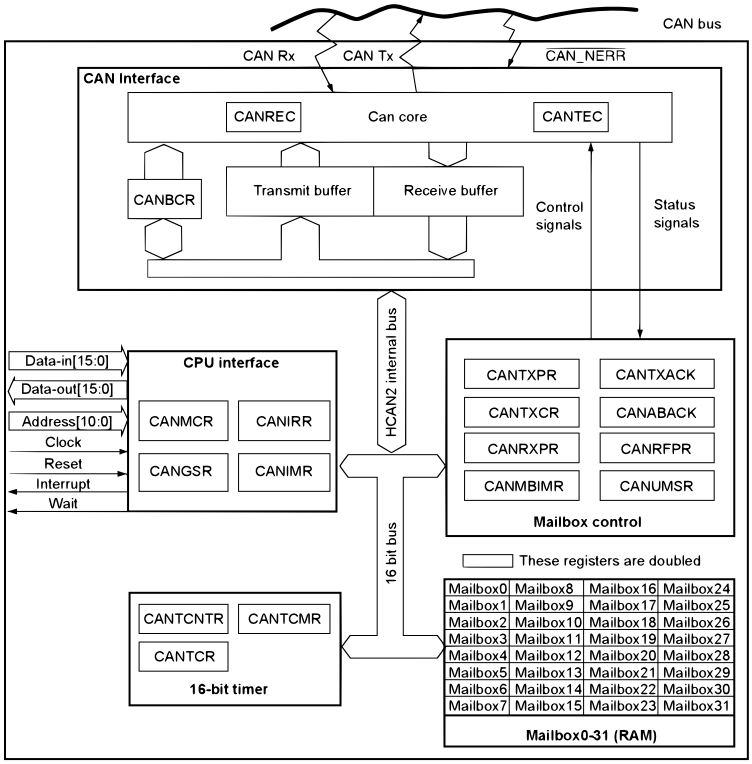
4.2 Procesor SH7760

Součástí modulu je 32-bitový RISC procesor Renesas SuperH 7760 (viz. [4]) s 16-bitovou instrukční sadou operující na frekvenci 200 MHz. K procesoru jsou připojeny dvě cache: instrukční (16KB) a operandová (32KB), a synchronní operační pamět DRAM o velikosti 64MB s frekvencí přístupu 66 MHz. V čipu jsou zapouzdřeny kromě CPU i řadič HCAN2 pro komunikaci po CAN sběrnici a jiné obvody pro vstup/výstup. Modul SH7760 se připojuje k základní desce EXM32 pomocí patice s nulovou silou, do které je upevněn pomocí několika šroubů, které brání ztrátě kontaktu mezi procesorovým modulem a základní deskou.

4.3 Řadič HCAN2

HCAN2 (viz. [4]) je řadič vyvíjený společnosti Renesas pro komunikaci po CAN sběrnici. Řadič obsahuje dva interface pro připojení ke sběrnici, oba jsou identické, odlišující se pouze paměťovým prostorem kam jsou namapovány. Řadič je kompatibilní se standardem CAN 2.0A a CAN 2.0B a taktéž s normou ISO 11898 [5]. Řadič má 32 komunikačních objektů (*Mailboxů*), pomocí nichž komunikuje po sběrnici. Podporuje komunikační rychlost až 1 Mbps, pro přenos zpráv používá interní mechanizmus prioritizace.

4.3.1 Struktura řadiče



Obrázek 4.2: Blokové schema řadiče HCAN2 [4]

Řadič HCAN2 je možné rozdělit do pěti bloků:

- **MicroProcessor Interface:** Umožňuje komunikaci mezi procesorem a vnitřními registry řadiče. *MPI* také obsahuje logiku pro probuzení ze *SLEEP* režimu při zjištění aktivity na CAN sběrnici.
- **Mailbox:** RAM pamět se všemi *mailboxy*, popis struktury jednotlivých *mailboxů* je popsán v 4.3.2.
- **Mailbox Control:** Tento blok při příjmu zpráv kontroluje identifikátor zprávy proti masce a identifikátoru jednotlivých *mailboxů* nastavených pro příjem a nastavuje příslušné příznaky, při odesílání vybírá správný objekt k odeslání (dle priority) a nastavuje příznaky. V neposlední řadě řídí přístup k RAM paměti *mailboxů*, jelikož do této paměti přistupuje jak CPU, tak samotný blok *Mailbox Control*.
- **Timer:** Tento blok obsahuje 16-bitový volně běžící časovač ovládaný CPU. Obsahuje Time Compare Match Register, ve kterém je uložena hodnota, při jejímž

dosažení vyvolá časovač přerušení. Hodinový signál je odvozen od systémového hodinového signálu.

- **CAN Interface:** Tento blok je fyzicky připojen ke CAN sběrnici a obstarává vlastní komunikaci po sběrnici podle CAN specifikace 2.0. Blok také obsahuje *Transmit* a *Receive Error Counter* (čítače chyb), registry pro nastavení rychlosti komunikace (*BCR0,1*) a slouží jako dočasné úložiště přijatých nebo odesílaných zpráv.

4.3.2 Struktura komunikačního objektu - mailboxu

Address	Data bus																Access Size	Field Name
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
H'100 + N*32	0	STDID[10:0]											RTR	IDE	EXTID [17:16]		16 bits	Control
H'102 + N*32	EXTID[15:0]																16 bits	
H'104 + N*32			NMC	ATX	DART	MBC[2:0]			0		CBE		DLC[3:0]				8/16 bits	
H'106 + N*32	TimeStamp[15:0]																16 bits	Time stamp
H'108 + N*32	MSG_DATA_0 (first Rx/Tx byte)								MSG_DATA_1								8/16 bits	Data
H'10A + N*32	MSG_DATA_2								MSG_DATA_3								8/16 bits	
H'10C + N*32	MSG_DATA_4								MSG_DATA_5								8/16 bits	
H'10E + N*32	MSG_DATA_6								MSG_DATA_7								8/16 bits	
H'110 + N*32	Local acceptance filter mask 0 (LAFM0)																16 bits	LAFM
H'112 + N*32	Local acceptance filter mask 1 (LAFM1)																16 bits	

Obrázek 4.3: Struktura mailboxu [4]

Význam jednotlivých polí:
STDID[10:0] ***Standard ID*:** horních 11 bitů identifikátoru zprávy
EXTID[17:16] a EXTID[15:0] ***Extended ID*:** při IDE=1 obsahuje rozšířenou část identifikátoru
RTR ***Remote Transfer Request*:** typ zprávy, 0 - *Data Frame*, 1 - *Remote Frame*
IDE ***Identifier Extension*:** příznak, zda-li zpráva používá standardní nebo rozšířený identifikátor, 0 - standardní, 1 - rozšířený
NMC ***New Message Control*:** postup při přichodu zprávy do *mailboxu*, ve kterém je uložena nepřečtená zpráva, 0 - *mailbox* se přeskočí a pokračuje se ve výběru vhodného

mailboxu pomocí masky a identifikátoru zprávy, 1 - nově příchozí zpráva je uložena do *mailboxu* a přepíše tak původní

ATX *Automatic Transmition of Data Frame*: reakce *mailboxu* na přijetí *Remote Request Frame*, 0 - pouze se nastaví příznak v *CANRFPR*, 1 - automaticky se odešle zpráva uložena v *mailboxu* a příznak se nenastavuje

DART *Disable Automatic Re-Transmition*: postup při detekci chyby při odesílání zprávy, 0 - zpráva není znovu odeslána, 1 - řadič se pokusí o znovuodeslání zprávy

MBC *Mailbox Configuration*: nastavení *mailboxu*

CBE *CAN Bus Error*: Při použití externího CAN vysílače/přijímače je v tomto bitu uložen příznak, zda byla zpráva bezchybně odeslána/přijata

DLC *Data Length Code*: počet datových bytů, 0b000 odpovídá nulové délce, 0b1xxx odpovídá délce 8 bytů

TimeStamp[15:0] *Time Stamp*: časové razítko zprávy, zpravidla obsahuje informaci o času odeslání

MSG_DATAx *Data Byte x*: jendotlivé datové byty zprávy

LAFM *Local Acceptance Filter Mask*: maska použítá při hledání vhodného *Mailboxu* pro uložení přijaté zprávy, pro odesílanou zprávu nemá toto pole význam

4.3.2.1 Maska příchozích zpráv

H'110 + N*32	0	STDID_LAFM[10:0]	0	0	EXTID_LAFM [17:16]	16 bits	LAFM field
H'112 + N*32	EXTID_LAFM[15:0]					16 bits	

Obrázek 4.4: Pole LAFM v paměti mailboxu [4]

Při přijetí zprávy se projdou *mailboxy* počínaje 31 a konče 0. U každého se kontroluje nastavení *mailboxu* (zdali je nastaven pro příjem tohoto typu zprávy) a *mailboxů*, které vyhovují nastavením se zkontroluje, zdali souhlasí identifikátor *mailboxu* a zprávy. LAFM *Local Acceptance Filter Mask* je bitová maska filtru. Pro jednodužší porovnání má maska stejné členění v paměti jako identifikátor. Proto stačí pouze na správném místě nastavit hodnotu 1 pro ignorování tohoto bitu v identifikátoru nebo 0 pokud má být onen bit shodný jak v identifikátoru *mailboxu* tak v identifikátoru zprávy.

4.4 Ostatní hardware

4.4.1 PC104 s řadičem CAN SJA1000

Tento průmyslový počítač jsem využil jako zařízení pro otestování komunikace po CAN sběrnici, jelikož je vybaven modulem s řadičem Philips SJA1000 umožňující připojit zařízení ke 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, vyrovnávací paměť 128 MB, LPT paralelní port, čtyři COM sériové porty, šíťovým adaptérem 10/100 Mbps, vstup pro PS/2 myš a klávesnici, výstup na CRT monitor a konektor pro CompactFlash kartu, ze které bootuje do systému. Na tomto počítači běží OS Linux (jádro 2.6.20 s podporou *Real-Time*) do kterého je jako modul přidán driver LinCAN, který obsluhuje řadič Philips SJA1000.

4.4.2 CAN Analyzer - KVASER

CAN Analyzer je přídavná PCMCIA karta do notebooku vyráběná společností KVASER. Na kartě jsou dva Philips SJA1000 řadiče pro připojení ke CAN sběrnici. Karta podporuje Real-Time komunikaci po CAN sběrnici. Dodávaný software umožňuje jak monitoring sběrnice, tak i aktivní zapojení do komunikace. Karta byla použita k monitoringu CAN sběrnice a generování jednoduchých zpráv, které byly přijímány paralerně jak PC104 tak řadičem HCAN2.

Kapitola 5

Příprava EXM32-SH7760 s řadičem HCAN2

Na systému EXM32-SH7760 běží operační systém Linux, který je zaváděn při bootu z karty *Compact Flash*. Proces zavádění operačního systému je relativně složitý. Po zapnutí *EXM32-SH7760* je nastaven čítač instrukcí do adresního prostoru Flash paměti. Proto je proces zavádění systému rozdělen do dvou částí. První část, kterou obstarává *bootloader IPL*, se skládá z nastavení základních hodnot procesoru, inicializace karty *Compact Flash* a zavolaní zavadeče LILO z *boot sectoru* na CF kartě. V druhé části přebírá řízení LILO, který zavede jádro systému z CF karty. Problémem bylo vytvoření *boot sectoru* na CF kartě. To nebylo možné ze systému na EXM32, protože nemohl být zaveden. Bylo tedy nutné *boot sector* vytvořit pomocí jiného počítače. Prostým zavoláním LILO na hostitelském počítači by se dostal do zaváděcího sektoru CF karty binární kód pro x86 architekturu. Proto byl zkompilován binární obraz zavaděče pro SH7760 a uložen na CF kartu, odkud ho LILO spustěné na hostitelském počítači nakopíruje spolu s obrazem jádra systému Linux do *boot sectoru* na CF kartě.

5.1 Cross-Kompilace

Cross-kompilace je proces, kdy se překládá na jednom typu architektury pro jiný typ. Výsledkem takovéhoto překladu je potom binární soubor, který je spustitelný na cílové architektuře. Jelikož architektura SH7760 není přímo kompatibilní s architekturou Intel x86, provádí se Cross-kompilace na běžném PC pro architekturu *SuperH 7760*. K tomuto

kroku jsou použity nástroje následujících verzí:

binutils-sh-linux: 2.6.1-2

gcc-sh-linux: 3.4.4-2

gdb-sh-linux: 20071217-5

glibc-sh-linux: 2.3.3-28.12

slibc-sh4-linux: 2.3.3-28.12

Soubory jsou stažitelné z internetu [7]. Balíčky bylo nutné konvertovat pro distribuci Debian pomocí nástroje *Alien*, který provede potřebný převod. Výsledné balíčky jsou přiloženy na CD.

5.2 Nové jádro 2.6.21

Vývoj softwaru by měl probíhat na co možná nejnovější verzi hardwaru a softwaru. Pro vývoj ovladače je tento požadavek zvláště kritický, proto bylo rozhodnutou aktualizovat jádro. Bylo použito nejnovější jádro Vanilla Kernel 2.6.21-3, volně stažitelné z internetu (např. [8]). Poté byl aplikován patch vydaný výrobcem modulu EXM32-SH7760 pro jádro verze 2.6.21 pomocí programu *quilt*.

5.2.1 Cross-Kompilace a výchozí nastavení

Po rozbalení zdrojů jádra a updatování pomocí patche byla vytvořena konfigurace jádra pomocí výchozích parametrů pro SH7760 zavoláním *make O=_build/sh4 ARCH=sh CROSS_COMPILE=sh4-linux- exm7760_defconfig*. Parametr *O=_build/sh4* slouží k definici cesty, kam se budou ukládat dočasné a přeložené soubory, v tomto případě do podadresářů _build/sh4 v adresáři jádra. Parametr *ARCH=sh* definuje architekturu pro kterou se jádro překládá, to umožňuje *make* zvolit správné verze hlavičkových souborů. *CROSS_COMPILE=sh4-linux-* určuje jaký prefix mají nástroje použité pro překlad (GCC, LD...). Parametrem *exm7760_defconfig* volí uživatel, jakou výchozí konfiguraci má *make* použít.

5.2.2 Změna boot sektoru - LILO

Po překladu je nutné jádro nahrát na kartu *Compact Flash* do adresáře */boot/*, příslušným způsobem změnit lilo.conf a zavolat *lilo* aby pozměnil boot sektor na disku. Při prvním spuštění se zastavil boot proces při připojování *root file systemu* s chybou *Kernel Panic*, proto veškeré další pokusy o zprovoznění nového jádra musely využívat čtečku karet pro přenos jádra na paměťovou kartu. Taktéž bylo nutné volat *lilo* se speciálními parametry *-r /mnt/flash -C /lilo.conf*, který umožnil přepsat *boot sector* CF karty připojené pomocí *mount* příkazu.

5.2.3 Přenastavení parametrů jádra v lilo.conf

Výsledkem několika hodin pokusů s konfigurací jádra byl kompromis, při kterém je jádro načteno z karty *Compact Flash*, ale *root file system* je připojen ze sítě. Toho bylo dosaženo pomocí přidání parametrů *console=ttySC1,115200 root=/dev/nfs nfsroot=192.168.1.244:/exm32 ip=192.168.1.245:192.168.1.244:192.168.1.244: 255.255.255.0:exm32* do souboru *lilo.conf*. První parametr *console=ttySC1,115200* zapíná logování do sériového portu s rychlostí přenosu 115200 kbps. Jelikož EXM32 s SH7760 nemá nastavený grafický výstup pro CRT monitor, je sériový port jediná možnost, jak sledovat průběh bootování systému. Parametry *root=/dev/nfs nfsroot=192.168.1.244:/exm32* definují jádru, že *root file system* se připojí přes NFS z počítače s IP adresou *192.168.1.244*, který má nastavený export na adresář */exm32*. Poslední parametr *ip=192.168.1.245: 192.168.1.244:192.168.1.244:255.255.255.0:exm32* určuje IP adresu počítače EXM32, výchozí bránu, hlavní DNS server a DNS název počítače EXM32. Toto řešení je výhodnější pro vývoj ovladače, neboť není třeba přenášet přeložený modul přes síť do zařízení a jednoduše se nakopíruje do některého podadresáře exportovaného */exm32*.

Kapitola 6

Implementace driveru LinCAN pro řadič HCAN2

6.1 Popis

Přidání podpory nového hardwaru do ovladače *LinCAN* je zdánlivě jednoduchá činnost, ovšem při hlubším zkoumání dojdeme k závěru, že je třeba nejdříve proniknout do vnitřní struktury driveru a správně celou implementaci navrhnout. Jelikož je *LinCAN* ovladač i pro *Real-Time* operační systémy, je kladen veliký důraz na rychlost, zejména pak v často volaných funkcích.

Zdrojové kódy *LinCANu* obsahují mimo jiné i soubory *include/template.h* a *src/template.c*. Tyto soubory, jak název napovídá, jsou šablony pro přidání karty. Soubor *include/template.h* obsahuje hlavičky funkcí, jejichž těla najdeme v souboru *src/template.c*. Tyto soubory jsou použity jako základ přidávané karty *SH7760*. Driver nyní obsahuje tři kontrolery (*Intel i82527, Philips 82c200* a *Philips SJA1000*). Jelikož kontroler se do struktury driveru *LinCAN* nepřidává často, neobsahuje struktura zdrojů driveru *LinCAN* pro to žádné šablony. Proto byl použit kód pro kontroler SJA1000 jako předloha pro řadič HCAN2. Kontroler SJA1000 má sice jinou vnitřní strukturu než řadič HCAN2, ale jeho funkce jsou popsány v dokumentaci k driveru *LinCAN*, proto je snadnější dohledat, co která funkce dělá.

Kromě vytvoření nových souborů s hlavičkami a těly funkcí je ještě třeba upravit některé soubory (např. *config.omk*), aby bylo možné zakompilovat nové soubory do ovladače. Stručný popis jednotlivých modifikovaných souborů a jejich obsahu naleznete v následující sekci.

6.2 Implementace

6.2.1 Změny v config.omk a config.omk-default

LinCAN využívá pro kompilaci Ocera Make Framework. Při zavolání příkazu *make* v kořenovém adresáři driveru se ze standardního *MakeFile* zavolá *MakeFile.omk*, který obstará celou kompilaci ovladače včetně malých *Utility* aplikací, jako je *sendburst, readburst* apod. Hlavním konfiguračním souborem je *config.omk* jehož předlohou je *config.omk-default*. Do těchto dvou souborů jsem přidal řádek umožňující zakompilovat nově přidanou *kartu* a *kontroler* do driveru.

```
CONFIG_OC_LINCAN_CARD_sh7760=n
```

6.2.2 Změny v src/boardlist.c

Soubor *boardlist.c* obsahuje pole sktruktur *boardtype_t*, které se plní při kompilaci podle definovaných konstant v souboru *config.omk*. Do souboru *boardlist.c* bylo třeba nadefinovat hlavičku funkce pro registraci kontroleru HCAN2

```
extern int sh7760_register(struct hwspecops_t *hwspecops);
```

a do pole přidat položku pro kontroler HCAN2 na kartě SH7760.

```
#if defined(CONFIG_OC_LINCAN_CARD_sh7760)
{"sh7760", sh7760_register, 1},
#endif
```

Pokud je v konfiguračním souboru *config.omk* definovaná konstanta *CONFIG_OC_LINCAN_CARD_sh7760=y*, je podmínka vyhodnocena překladačem jako splněná a položka je přidána do pole.

6.2.3 Obsah include/sh7760.h

Soubor obsahuje pouze hlavičky funkcí a definice konstant používaných v souboru *src/sh7760.c*.

#define NR_82527 0: Na kartě není žádný kontroler Intel 82527.

#define NR_SJA100 0: Na kartě není žádný kontroler Philips SJA1000.

#define NR_ALL 2: Na kartě jsou dva kontrolery HCAN2

#define SH7760_CAN_IRQ 56: Číslo přerušení, které kontroler používá.

#define SH7760_CAN_CHIP_OFFSET 0x10000: Offset mapování kontrolerů v paměti.

#define SH7760_CAN_CLOCK 33300000: Frekvence hodin kontroleru, používá se pro výpočet hodnoty *Bit rate*.

#define IO_RANGE 0x10000: Velikost paměti rezervované pro jeden kotroler.

6.2.4 Obsah include/hcan2.h

Tento soubor obsahuje hlavičky funkcí implementovaných v *src/hcan2.c* a jsou v něm nadefinovány adresy jednotlivých registrů kontroleru. Jelikož přístup do paměti kontroleru musí být až na vyjimečné případy výhradně 16-ti bitový, jsou v tomto hlavičkovém souboru napsány i následující dvě funkce:

inline uint16_t **can_read_reg_w** (*const struct canchip_t *pchip, unsigned reg*)

Tato funkce slouží k zápisu 16-ti bitové hodnoty do registru kontroleru.

inline void **can_write_reg_w** (*const struct canchip_t *pchip, unsigned reg*)

Tato funkce slouží k čtení 16-ti bitové hodnoty z registru kontroleru.

6.2.5 Obsah src/sh7760.c

Funkce obsažené souborem *sh7760.c* jsou volány při inicializaci karty během přidávání modulu do systému. Pokud není uvedeno jinak, vrací popisovaná funkce v případě chyby zápornou hodnotu a pokud proběhne bezchybně hodnotu 0.

int **sh7760_request_io** (*struct candevice_t *candev*)

Funkce požádá systém o přidělení přístupu do paměti kontrolerů HCAN2. Funkce předpokládá, že mapování paměťového prostoru kontrolerů do paměti je souvislé.

int **sh7760_release_io** (*struct candevice_t *candev*)

Funkce uvolňuje paměť přidělenou driveru pro kartu SH7760. Funkce nemůže selhat a proto je návratová hodnota vždy 0.

int **sh7760_reset** (*struct candevice_t *candev*)

Funkce volá resetovací rutiny kontrolerů.

int **sh7760_init_hw_data** (*struct candevice_t *candev*)

Funkce nastavuje výchozí hodnoty struktury *candevice_t* (viz. [2] kap. 1.6), ve které jsou uloženy důležité parametry karty SH7760. Návratová hodnota je vždy 0.

int **sh7760_init_chip_data** (*struct candevice_t *candev, int chipnr*)

Funkce volá rutinu příslušného kontroleru pro incializaci kontroleru (*hcan2_fill_chipspecops*) a nastavuje hodnoty struktury *can_chip_t* (viz. [2] kap. 1.6), ve které jsou uložena věškerá data kontroleru. Návratová hodnota je vždy 0.

int **sh7760_init_obj_data** (*struct canchip_t *chip, int objnr*)

Funkce volá inicializační rutinu příslušného komuničního objektu *mailboxu* (dle parametru *objpnr*) a nastavuje hosnoty struktury *can_chip_t* (viz. [2] kap. 1.6), ve které jsou uloženy parametry *mailboxu*. Návratová hodnota je vždy 0.

int **sh7760_program_irq** (*struct candevice_t *candev*)

Funkce slouží pro nastavení programovaného přerušení a *není implementována*.

void **sh7760_write_register** (*unsigned data, can_ioptr_t address*)

Funkce zapíše hodnotu předanou jako parametr do paměti kontroleru s offsetem *address* od základní adresy kontroleru.

unsigned **sh7760_read_register** (*can_ioptr_t address*)

Funkce vrací hodnotu načtenou z paměti kontroleru s offsetem *address* od základní adresy kontroleru.

6.2.6 Obsah src/hcan2.c

Tento soubor obsahuje obslužné rutiny pro kontroler HCAN2. Některé funkce nejsou implementovány, protože to dosavadní zjištění nevyžadovala. Pokud není uvedeno jinak,

vrací popisovaná funkce v případě chyby zápornou hodnotu a pokud proběhne bezchybně, vrací hodnotu 0.

int **hcan2_chip_config** (*struct canchip_t *chip*)

Funkce slouži k nastavení masky přerušení a komunikační rychlosti při kontroleru přepnutém do *reřimu konfigurace*.

int **hcan2_enable_configuration** (*struct canchip_t *chip*)

Funkce zapne *režim konfigurace* tím, že vydá kontroleru povel, aby přešel do *HALT* módu. Před přechodem do *HALT* módu kontroler dokončí probíhající odesílání či příjem zprávy. V režimu *HALT* se kontroler neúčastní komunikace po sběrnici CAN.

int **hcan2_disable_configuration** (*struct canchip_t *chip*)

Funkce přepne kontroler zpět z *HALT* módu do pracovního módu. Kontroler se musí nejprve synchronizovat se sběrnicí, proto čeká na 11 bitů recesivní úrovně, než se zapojí do komunikace po sběrnici. Přepnutím módu se zruší *režim konfigurace*.

int **hcan2_baud_rate** (*struct canchip_t *chip, int rate, int clock, int sjw, int sampl_pt, int flags*)

Funkce spočítá správné hodnoty registrů pro nastavení rychlosti komunikace, které následně uloží do paměti kontroleru zavoláním funkce *hcan2_set_btregs*. Pokud není možné spočítat vhodné hodnoty pro nastavení komunikační rychlosti (chyba rychlosti vypočítané by byla větší než 10%), vrací funkce zápornou hodnotu.

int **hcan2_set_btregs** (*struct canchip_t *chip, unsigned short btr0, unsigned short btr1*)

Funkce nejprve upraví hodnoty *btr0 a btr1* maskou. Maska zaručí, že se do rezervovaných bitů registrů neuloží chybné hodnoty. Hodnoty upravené maskou se zapíší do paměti kontroleru. Funkce nemůže selhat, její návratová hodnota je proto vždy 0.

int **hcan2_start_chip** (*struct canchip_t *chip*)

Funkce vypne *režim konfigurace*, kontroler je poté schopen komunikovat po sběrnici.

int **hcan2_stop_chip** (*struct canchip_t *chip*)

Funkce zapne *režim konfigurace*, kontroler se neučastní komunikace po sběrnici.

int **hcan2_attach_to_chip** (*struct canchip_t *chip*)

Funkce resetuje kontroler (zavoláním *hcan2_reset_chip*). Aby byly zaručeny výchozí nastavení registrů, zavolá funkci *hcan2_chip_config*, která nastaví výchozí hodnoty. Funkce *hcan2_attach_to_chip* je volána v okamžiku připojení ke kontroleru (např při otevření souboru zařízení */dev/canX*).

int **hcan2_release_chip** (*struct canchip_t *chip*)

Funkce ukončí zpracování zpráv kontrolerem. Obvykle je tato funkce volána při zavření souboru zařízení */dev/canX*.

int **hcan2_standard_mask** (*struct canchip_t *chip, unsigned short code, unsigned short mask*)

Funkce nastaví standardní identifikátor a LAF 4.3.2.1.

int **hcan2_extended_mask** (*struct canchip_t *chip, unsigned long code, unsigned long mask*)

Funkce nastaví rozšířený identifikátor a LAF 4.3.2.1.

int **hcan2_message15_mask** (*int irq, struct canchip_t *chip*)

Funkce nastavuje masku pro patnáctý komunikační objekt. Tato funkce není implementována.

int **hcan2_pre_read_config** (*struct canchip_t *chip, struct msgobj_t *obj*)

Funkce volá interní rutinu obsluhy kontroleru *hcan2_setup_mbox4read*, která připraví komunikační objekt na příjem zprávy.

int **hcan2_pre_write_config** (*struct canchip_t *chip, struct msgobj_t *obj, struct canmsg_t *msg*)

Funkce nejdříve vynuluje příznaky přerušení a zavolá interní rutinu obsluhy kontroleru *hcan2_setup_mbox4write*, která připraví komunikační objekt na odeslání zprávy.

int **hcan2_send_msg** (*struct canchip_t *chip, struct msgobj_t *obj, struct canmsg_t *msg*)

Funkce spustí vysílání a počká na dokončení odesílání zprávy.

int **hcan2_remote_request** (*struct canchip_t *chip, struct msgobj_t *obj*)

Funkce se používá pro odeslání zprávy s žádostí o data (*Remote Frame Request*). Tato funkce není implementována.

int **hcan2_irq_handler** (*int irq, struct canchip_t *chip*)

Tato funkce, volaná jádrem driveru, obsluhuje přerušení. Na základě příznaku v *registru přerušení* zavolá funkci pro přečtení zprávy *hcan2_irq_read_handler* nebo funkci pro odeslání zprávy *hcan2_irq_write_handler*.

int **hcan2_irq_accept** (*int irq, struct canchip_t *chip*)

Funkce není implementována.

int **hcan2_config_irqs** (*struct canchip_t *chip, short irqs*)

Funkce maže příznaky a nastavuje masku přerušení.

int **hcan2_clear_objects** (*struct canchip_t *chip*)

Funkce vymaže data a nastavení z komunikačních objektů. Postupně se volá rutina *hcan2_clear_mbox* pro všechny komunikační objekty. Návratová hodnota je vždy 0.

int **hcan2_check_tx_stat** (*struct canchip_t *chip*)

Funkce slouží ke kontrole stavu odesílání zprávy. Návratová hodnota je 0 v případě, že zpráva je již odeslána nebo funkce vrací zápornou hodnotu, pokud se odeslání nepodařilo nebo stále probíhá.

int **hcan2_wakeup_tx** (*struct canchip_t *chip, struct msgobj_t *obj*)

Funkce zkontroluje frontu zpráv k odeslání a pokud je připravena zpráva, zavolá *hcan2_irq_write_handler*. Návratová hodnota je vždy 0, pokud funkce není volána s parametrem mailboxu 0, ten smí sloužit pouze k příjímání zpráv, nikoliv odesílání.

int **hcan2_filtch_rq** (*struct canchip_t *chip*)

Funkce je volána jádrem droveru při každém otevření *FileDescriptoru /dev/canX*, volá funkci *hcan2_pre_read_config* a vrací její návratovou hodnotu.

int **hcan2_register** (*struct chipspecops_t *chipspecops*)

Funkce nastaví ukazatele na funkce volané jádrem driveru nebo obsluhou karty SH7760. Návratová hodnota je vždy 0.

int **hcan2_fill_chipspecops** (*struct canchip_t *chip*)

Funkce nastaví jméno kontroleru, počet komunikačních objektů a zavolá *hcan2_register*. Návratová hodnota je vždy 0.

int **hcan2_reset_chip** (*struct canchip_t *chip*)

Funkce vyvolá restart kontroleru a nastaví výchozí hodnoty konfigurace kontroleru.

void **hcan2_irq_read_handler** (*struct canchip_t *chip, struct msgobj_t *obj*)

Funkce načte přijatou zprávu z paměti komunikačního objektu do FIFO fronty příchozích zpráv v driveru *LinCAN*.

void **hcan2_irq_write_handler** (*struct canchip_t *chip, struct msgobj_t *obj)*)

Funkce odešle zprávu zavoláním rutin *hcan2_pre_write_config* a *hcan2_send_msg*.

void **hcan2_clear_irq_flags** (*struct canchip_t *chip*)

Funkce vynuluje všechny příznaky přesušení.

void **hcan2_clear_mbox** (*struct canchip_t *chip, int msgobj_idx*)

Funkce nastaví v paměti příslušného komunikačního objektu výchozí hodnoty.

void **hcan2_setup_mbox4write** (*struct msgobj_t * obj, struct canmsg_t * msg*)

Funkce uloží hodnoty z *msg* do paměti komunikačního objektu *obj* a připraví objekt na odeslání dat.

void **hcan2_setup_mbox4read** (*struct msgobj_t * obj*)

Funkce nastaví identifikátor a LAF 4.3.2.1. Jelikož v okamžiku volání této funkce není znám identifikáro přijímané zprávy, nastaví funkce identifikátor na hodnotu 2048 a masku tak, aby mailbox přijímal všechny zprávy s rozšířeným identifikátorem.

Kapitola 7

Závěr

Po mnoha pokusech se podařilo Cross-kompilované jádro zavést do počítače EXM32-SH7760. Jádro připojuje *root file system* přes síťové rozhraní pomocí protokolu nfs. Což se ukázalo jako výhodné pro vyvíjení ovladače, bohužel, mělo-li by se jednat o trvalé řešení, je toto zcela nevhodné. Při každém zavádění systému je totiž nutné mít v siťovém okolí přítomný počítač s *file systémem* pro počítač EXM32/SH7760.

Během vývoje driveru docházelo, z počátku ojediněle, ke konci však již často, ke zkoušení ovladače. Bylo ověřeno úspěšné přidání driveru do systému *LinCAN*. Také bylo ověřeno správné nastavování komunikačních objektů před i po příjmu či vysílání a komunikace pomocí několika náhodných mailboxů na obou kontrolerech procesoru SH7760. Pro verifikaci přijímání a odesílání zpráv bylo použito jednoduché zapojení sběrnice CAN se třemi uzly: průmyslový počítač EXM32-SH7760 s kontrolerem HCAN2, průmyslový počítač PC104 s kontrolerem Philips SJA1000 a CAN Analyzer KVASER také s kontrolerem Philips SJA1000. Na obou průmyslových počítačích byl použit pro odesílání zpráv program *sendburst* a pro příjem zpráv program *readburst*. Oba tyto programy jsou součástí ovladače *LinCAN*. Na laptopu s CAN Analyzerem KVASER se pro příjem i odesílání zpráv používala aplikace pro MS Windows *CANKing* dodávaná spolu s CAN Analyzerem.

Při verifikaci byl kladen důraz na správnou interpretaci těla zprávy (identifikátor, datové byty) a na rychlost obsluhy komunikace kontrolerem HCAN2. Komunikační rychlost sběrnice byla nastavena na 1000 kbps. Kontroler *HCAN2* v ovladači *LinCAN* komunikoval maximální rychlostí, jakou byly schopny kontrolery SJA1000 vyvinout, bez ztráty zprávy či jejím chybném rozkódování. Z toho je možné vyvodit závěr, že v daném případě byla potvrzena funkčnost ovladače.

Během testování bylo zjištěno, že obsluha kontroleru nedostane žádnou zprávu o

zavření *FileDescriptoru* uživatelskou aplikací. V důsledku toho zůstane používaný mailbox nastaven pro příjem zpráv. Vzhledem ke způsobu vyhodnocování odpovídajícího mailboxu při přijetí zprávy kontrolerem, má toto za následek v některých specifických případech ztrátu zprávy.

Literatura

[1] Doc. Ing. Petr Kocourek, CSc., Ing. Jiří Novák, Ph.D. *Přenos Informace*. ČVUT v Praze, 2006.

[2] Ing. Pavel Píša. *Linux/RT-Linux CAN Driver (LinCAN)*. (http://cmp.felk.cvut.cz/ pisa/can/doc/lincandoc-0.3.pdf), 2005.

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

[4] Renesas Technology Corp. *Hitachi SuperH RISC engine SH7760 - Hardware Manual*, 2003.

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

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

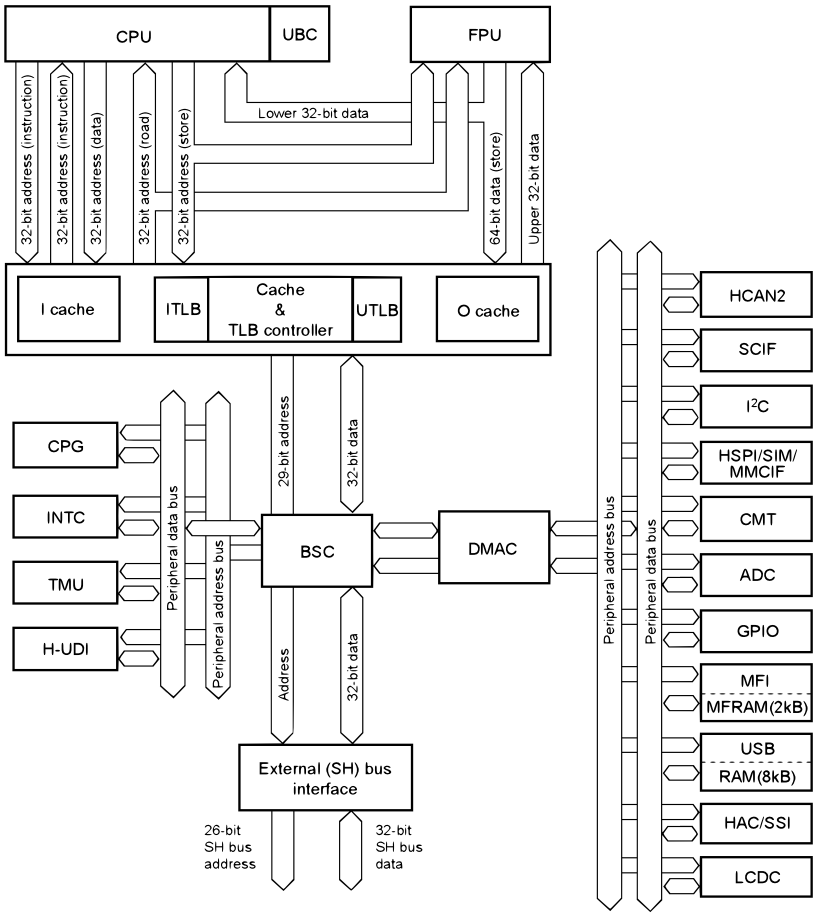
[7] SH-Linux. http://rpm.sh-linux.org/rpm-index-2004/.

[8] Linux Kernel. http://www.kernel.org.

Příloha A

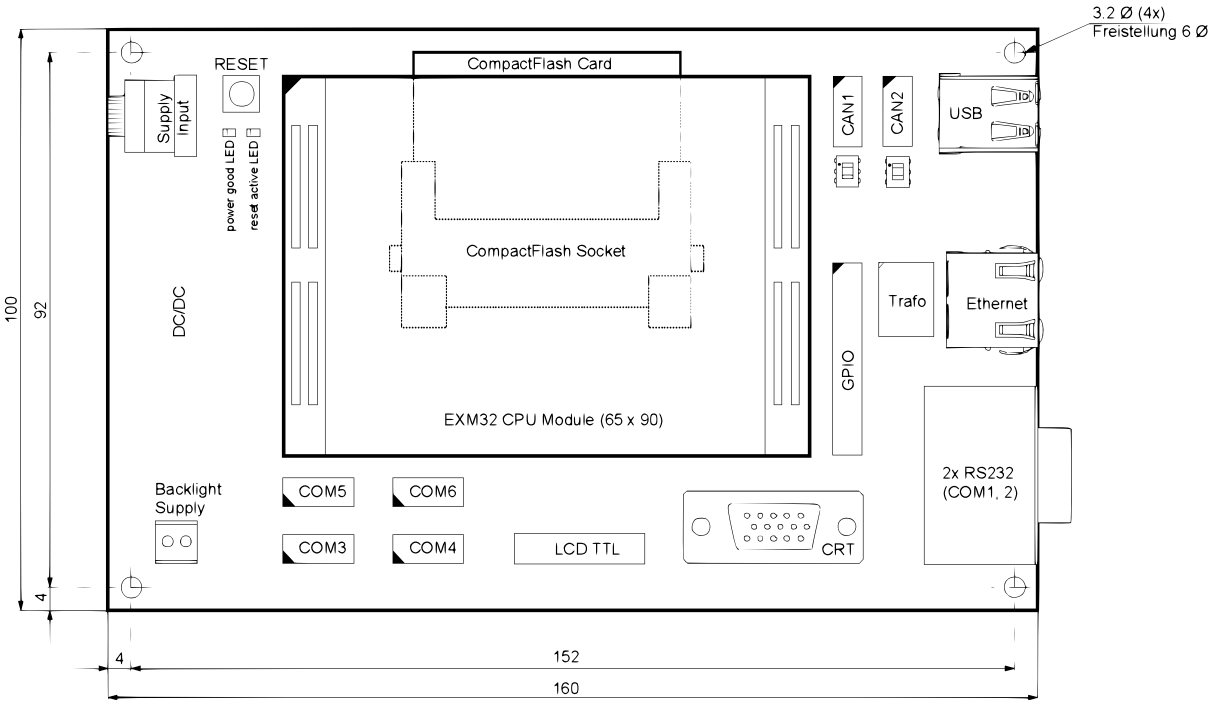
Schémata

A.1 Blokové schéma procesoru SH7760



Obrázek A.1: Blokové schéma SH7760 (převzato z [4])

A.2 Rozmístění konektorů na základní desce EXM32 MB Lite



Obrázek A.2: Rozmístění portů na desce (převzato z [4])

2 x CAN (IDC 10 pinů)
1 x USB (Typ A)
1 x Síťový konektor (RJ45)
6 x sériový interface (2 x DSUB 9 pinů, 4 x IDC 10 pinů)
Compact Flash zásuvka
1 x LCD (FFC/FPC 31 pinů, TFT LVTTL)
1 x VGA (DSUB 15 pinů)
napájení 12 V, souosý konektor
5 V/12 V napájení pro podsvícení LCD

Příloha B

Popis použití ovladače kontroleru HCAN2

B.1 Aplikace patche HCAN2 pro ovladač LinCAN

Přiložený soubor *lincan-0.3.4-pre2_HCAN2_patch.tar.gz* je třeba rozbalit do podadresáře *patches* ve struktuře zdrojů ovladače LinCAN. Po rozbalení archivu (např. *tar -xf lincan-0.3.4-pre2_HCAN2_patch.tar.gz* nebou použitím programu *Midnight Commander*) je nutné přidat do souboru *series* v adresáři *patches* řádek obsahující *sh4-hcan2*. Poté aplikovat patch zavoláním programu *quilt push* z kořenového adresáře ovladače LinCAN. *Quilt* je program pro správu patchů a je součástí distribuce OS Linux.

B.2 Nastavení kompilace ovladače LinCAN

Konfigurace kompilace je v souboru *config.omk*, pokud soubor neexistuje, je možné jej vytvořit zkopírováním *config.omk-default*. Předpokladem pro úspěšnou kompilaci ovladače jsou dostupné zdroje jádra systému běžící na počítači EXM32-SH7760. Cestu ke zdrojům jádra je nutné upravit v konfiguračním souboru například takto:

```
LINUX_DIR=/usr/src/exm32/linux-2.6.21.3/_build/sh4
```

Dále je třeba v konfiguračním souboru zaměnit řádek

```
CONFIG\_OC\_LINCAN\_CARD\_sh7760=n
```

za

```
CONFIG\_OC\_LINCAN\_CARD\_sh7760=y
```

Příklad konfigurace ovladače LinCAN je součástí přílohy.

B.3 Kompilace ovladače LinCAN a přidání modulu

Kompilace se spouší zavoláním *make* v kořenovém adresáři struktury ovladače LinCAN. Po kompilaci je modul umístěn v adresáři *_compiled/modules/lincan.ko*. Přidání modulu do systému je možné provést příkazem *insmod lincan.ko hw=sh7760 io=0xfe380000 irq=56,57 baudrate=1000*. Parametr *io=0xfe380000* nastavuje počáteční adresu, kam je namapován první kontroler HCAN2. Parametr *irq=56,57* udává přerušení používané kontrolery HCAN2 a *baudrate=1000* specifikuje rychlost komunikace po CAN sběrnici v kbps.

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.

BP

BP_2007_Martin_Petera.pdf: tato bakalářská práce ve formátu PDF

Documentation

CAN

Can20_specification.pdf: specifikace standardu CAN 2.0A a 2.0B

EXM32-SH7760

EXM32.pdf: příručka pro základní desku EXM32

EXM32_SH7760_CPU_Module.pdf: příručka pro procesorový modul SH7760

SH7760.pdf: kompletní dokumentace procesorového modulu SH7760

LinCAN

lincandoc-0.3.pdf: dokumentace k driveru LinCAN verze 0.3

EXM32-SH7760

Crosscompiler: adresář obsahuje Debian balíčky potřebné pro Cross-Kompilaci

Kernel

exm32-sh7760-2.6.21-070511_kernel.patch.bz2: patch pro jádro 2.6.21 dodávaný společností Renesas pro počítač EXM32-SH7760

linux-2.6.21.3.tar.bz2: zdroje použitého jádra Vanilla Kernel 2.6.21

.config: konfigurace jádra použitého při Cross-Kompilaci

Source

LinCAN

config.omk: příklad konfiguračního souboru ovladače LinCAN

lincan-0.3.4-pre2.tar.gz: zdrojové soubory LinCAN 0.3.4 pre-release 2

lincan-0.3.4-pre2_HCAN2_patch.tar.gz: patch pro LinCAN 0.3.4-pre2 obsahující ovladač pro kontroler HCAN2 obsažený v procesoru SH7760