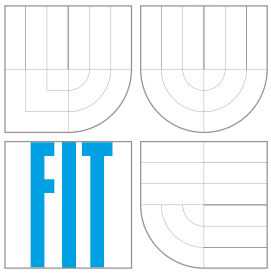
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A IMPLEMENTACE ŘADIČE SBĚRNICE CAN PRO VÝVOJOVOU DESKU ATMEL ATNGW100

DESIGN AND IMPLEMENTATION OF CAN DRIVER ON DEVELOPMENT BOARD ATMEL ATNGW100

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — KAREL SZURMAN
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci řadiče CAN sběrnice. Tato sběrnice je spolu s RTC modulem umístěna na vývojové desce Atmel ATNGW100. Zařízení komunikují pomocí SPI rozhraní. Programový řadič je implementován jako modul do jádra operačního systému Linux. Díky tomuto modulu jsou uživateli zpřístupněny rozhraní implementované sběrnice CAN a RTC modulu.

Abstract

This bachelor's thesis describe desing and implementation of CAN driver. This bus together with RTC module are located on development board Atmel ATNGW100. Devices communicates over SPI interface. Software driver is implemented as module for kernel of operating system Linux. Thanks to this module, interfaces of implemented CAN bus and RTC are accessed to user.

Klíčová slova

řadič, sběrnice CAN, vývojová deska, Atmel ATNGW100, vestavěný systém, Linux

Keywords

driver, CAN bus, development board, Atmel ATNGW100, embedded system, Linux

Citace

Karel Szurman: Návrh a implementace řadiče sběrnice CAN pro vývojovou desku Atmel ATNGW100, bakalářská práce, Brno, FIT VUT v Brně, 2009

Návrh a implementace řadiče sběrnice CAN pro vývojovou desku Atmel ATNGW100

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Další informace mi poskytli Ing. Martin Švéda a zaměstnanci firmy UNIS a.s. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Karel Szurman
22. května 2009

Poděkování

Rád bych zde poděkoval všem lidem, kteří mě pomáhali při psaní práce a hlavně při tvorbě její praktické části. V první řadě chci poděkovat mému vedoucímu Ing. Martinovi Švédovi z firmy UNIS a.s za velkou podporu, technické zázemí, nápady a cenné rady, které mi dal. Dále patří mé poděkování Ing. Václavu Šimkovi za vedení práce.

Obsah

1 Úvod **3**

2 Komunikační sběrnice CAN **4**
2.1 Obecně o CAN . . . 4
2.2 Fyzická vrstva sběrnice . . . 4
2.2.1 Komunikace na sběrnici . . . 4
2.2.2 Přenos dat . . . 5
2.3 Zabezpečení přenosu dat . . . 5
2.4 CAN zprávy . . . 6
2.4.1 Datová zpráva (Data Frame) . . . 6
2.4.2 Žádost o data (Remote Frame) . . . 6
2.4.3 Chybová zpráva (Error Frame) . . . 6
2.4.4 Zpráva o přetížení (Overload Frame) . . . 7
2.5 Architektura Controller Area Network . . . 7
2.6 Protokol CANAerospace . . . 7
2.6.1 Formát CAN zprávy . . . 8

3 Vestavěné systémy **9**
3.1 Embedded Linux . . . 10
3.2 Vytvoření cílového systému . . . 10

4 Vývojový kit Atmel ATNGW100 **11**
4.1 Architektura AVR32 . . . 11
4.1.1 Režimy procesoru . . . 11
4.1.2 Paměťová jednotka MMU . . . 12
4.1.3 Ochrana paměti pomocí MPU . . . 12
4.2 AVR32 Aplikační procesor . . . 12
4.2.1 Pipeline . . . 13
4.2.2 Paměti . . . 13
4.3 Periferie . . . 14
4.3.1 PIO piny . . . 14
4.3.2 SPI rozhraní . . . 14
4.3.3 Ostatní hardware . . . 14
4.4 Vestavěný operační systém Linux . . . 15
4.4.1 Souborový systém . . . 15
4.4.2 Zavaděč systému . . . 16
4.4.3 Úprava jádra . . . 17

5 Hardwarové řešení embedded systému **18**
5.1 Technický návrh zařízení 18
5.2 Fyzická implementace CAN sběrnice 18
5.2.1 CAN kontroler MCP2515 18
5.2.2 CAN Přijímač SN65HVD230 19
5.3 Modul RTC hodin 19

6 Jádro operačního systému **21**
6.1 User space a kernel space 21
6.2 Modulární přístup 21
6.3 Souborové operace 21

7 Analýza prostředků a návrh implementace řadiče sběrnice CAN **23**
7.1 Cross kompilace 23
7.2 Návrh řadiče 23

8 Implementace řadiče sběrnice CAN **25**
8.1 Modul řadiče 25
8.1.1 Inicializace hardware 25
8.1.2 Soubory zařízení 26
8.1.3 Zpracování CAN zpráv 26
8.2 Použití řadiče 26
8.2.1 Testovací software 26

9 Závěr **28**

A Obsah přiloženého CD **30**

B Schémata **31**

Kapitola 1

Úvod

CAN (Controller Area Network [7]) je rychlý sériový komunikační protokol s vysokou úrovní zabezpečení přenosu dat a bezporuchovým během, díky čemuž je jeho použití velmi vhodné pro kritické aplikace a pro vestavěné systémy, které kladou důraz na spolehlivost.

CAN sběrnice bude pro svou komunikaci používat jeden z vyšších protokolů, standard CANAerospace (viz. [14]) který byl vyvinut pro komunikaci ve vestavěných systémech v leteckých aplikacích.

Komunikační CAN sběrnice bude umístěna na externí desce spolu s RTC hodinami. Tento externí modul bude připojen k vývojovému kitu Atmel ATNGW100 [4], s jehož AVR32 procesorem bude probíhat komunikace pomocí SPI rozhraní. V tomto vestavěném zařízení bude nainstalován operační systém Linux. CAN sběrnice bude řízena pomocí řadiče, jehož návrh a implementace je cílem této práce.

Řadič bude zaveden do jádra operačního systému. Kde bude tvořit komunikační rozhraní mezi programovým vybavením vestavěného systému a zařízením, s kterým bude tento systém po CAN sběrnici komunikovat. V OS vestavěného zařízení bude nainstalován software, který bude komunikovat pomocí řadiče - posílat a přijímat data po CAN sběrnici. Získaná data, bude tento program zpracovávat. Pomocí tohoto vestavěného systému bude možné monitorovat jakékoliv průmyslové zařízení, schopné komunikace po CAN sběrnici. Vývojová deska Atmel ATNGW100 disponuje dvěma ethernetovými porty, díky kterým mohou být monitorovaná data ze sledovaného zařízení dále šířena např. na webové rozhraní.

Následující text obsahuje popis protokolu CAN sběrnice a standardu CANAerospace, pár slov o vestavěných systémech, úvod do architektury AVR32 procesorů, popis vývojové desky Atmel ATNGW100 a jeho vestavěného operačního systému, koncepci hardwarového řešení vestavěného zařízení a jádra operačního systému Linux. Poté následuje návrh koncepce implementace řadiče a samotného řešení s popisem praktického použití. Nad problematikou se zamýšlím a vše shrnují v závěru.

Kapitola 2

Komunikační sběrnice CAN

2.1 Obecně o CAN

Systém CAN sběrnice byl vyvinut firmou Bosch v roce 1986. Primárně byl tento sériový komunikační protokol vytvořen pro použití v automobilovém průmyslu, ale díky svým vlastnostem se jeho využití rychle rozšířilo do mnoha dalších odvětví použití. Díky vysoké rychlosti, velké míře zabezpečení přenosu dat, distribuovanému řízení systémů v reálném čase, bezporuchovému běhu, nízké cenně, snadnému použití se začal používat např. v průmyslové automatizaci, lékařské technice, transportních systémech nebo vestavěných zařízeních. Nyní je CAN - Controller Area Network protokol nejpoužívanější komunikační sběrnici ve vestavěných a strojních zařízeních [1].

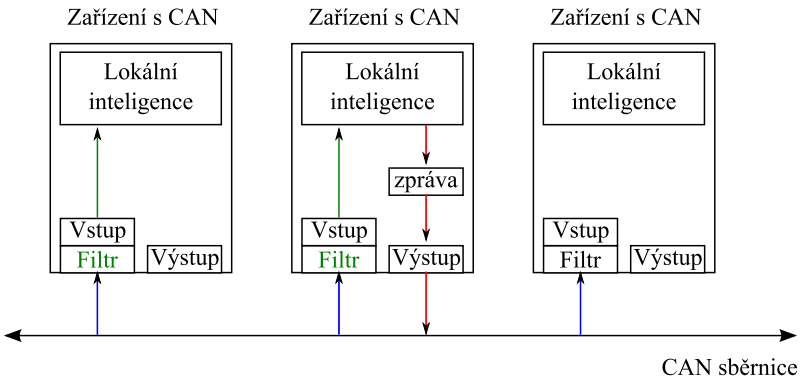
2.2 Fyzická vrstva sběrnice

CAN je protokol typu multi-master [13], tzn. že každý uzel (zařízení) na sběrnici může řídit chování jiného uzlu. Zařízení, které chce komunikovat začne vysílat data zprávy a v tom okamžiku se stane řídícím prvkem na sběrnici. Ostatní zařízení, která by chtěla komunikovat musí počkat až bude aktuální přenos dat dokončen.

2.2.1 Komunikace na sběrnici

Komunikace na sběrnici (viz. obr 2.1) je založena na prioritním systému zpráv. Při přijímání zpráv je v případě příchodu dvou zpráv na jeden uzel zpracována ta zpráva, která má vyšší prioritu. Přenášené zprávy neobsahují informace o cílovém zařízení, kterému jsou adresovány, a jsou implicitně přijímány všemi ostatními uzly na sběrnici. Každá zpráva obsahuje identifikátor, díky kterému je možné zajistit, aby odeslané zprávy byly přijaty pouze obsahem adresovaným zařízením. Každé zařízení může mít nastavený filtr zpráv, díky kterému lze provést takovouto selekci zpráv. Je nutné aby identifikátor zprávy byl v každém případě jedinečný, tato podmínka je dána specifikací CAN [7]. Zprávu ovšem může přijmout i více zařízení, pokud jsou nositeli daného identifikátoru.

V systémech využívajících CAN nejsou nijak využívány informace o konfiguraci sběrnice a připojených zařízeních, jako je např. již výše zmíněná adresa zařízení. Díky tomu mohou být do CAN systému velmi jednoduše přidány další zařízení, bez nutnosti provádět programové nebo hardwarové změny.



Obrázek 2.1: Komunikace zařízení na CAN sběrnici

2.2.2 Přenos dat

Při přenosu binárních dat CAN sběrnice používá NRZ (Non-Return to Zero, [3]) kódovaní, díky čemuž jsou přenášená data velmi odolná proti vnějšímu rušení. Při přenosu dat zakódovaných pomocí NRZ kódování je každý bit dán jedním časovým segmentem a po celou jeho dobu je signál na konstantní úrovni.

Datový nosič tvoří rozdílová linka, která se skládá ze dvou vodičů CAN_L a CAN_H. Logická hodnota bitu na sběrnici je dána logickým součinem AND napěťových úrovní na těchto vodičích. Sběrnice dosahuje přenosové rychlosti dat až 1 MBit/s při maximální délce datového vodiče 40 m. Tyto a jiné elektrické vlastnosti fyzické vrstvy jsou specifikovány normou ISO 11898.

2.3 Zabezpečení přenosu dat

Data jsou ve CAN zprávách při přenosu po sběrnici zabezpečena 5-ti různými způsoby, a to buď na úrovni zpráv nebo samotných bitů.

Zabezpečením na úrovni CAN zpráv je detekce chyb pomocí kontrolního součtu, neboli CRC (Cyclic Redundancy Check).

Dalším způsobem ochrany dat v CAN zprávách je kontrola jejich struktury, kdy jsou v těle zprávy na určitých pozicích umístěny kontrolní bity, jejichž správnost je ověřována.

Posledním způsobem zabezpečení na úrovni zpráv je ACK (Acknowledgment), což je způsob ověření, zda zpráva došla v pořádku. V případě že ano, pošle příjemce potvrzení o přijetí, což je kopie přijaté CAN zprávy, kde jsou v poli ACK bity nastaveny do dominantní úrovně. Pokud odesilatel zprávy neobdrží takovéto potvrzení, pokouší se odeslat zprávu znovu.

Na bitové úrovni se pro zabezpečení přenosu CAN zpráv používá kontrola shodnosti zprávy. Což znamená, že každý odesilatel zprávy dostane zpět kopii odeslané zprávy lišicí se v počátečních bitech a ACK poli.

Zbývající způsob zabezpečení je Bit Stuffing, který je založen na vkládání opačného bitu po každém bloku pěti stejných bitů. V případě porušení tohoto pravidla je vyvolána výjimka, která zpracuje tuto chybu (Stuff Error)

2.4 CAN zprávy

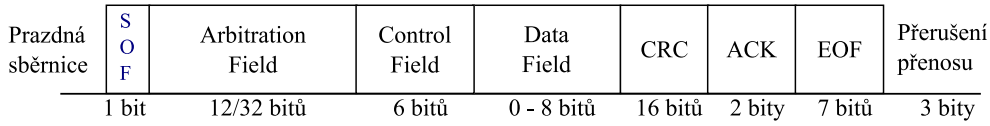
Mezi zařízeními, propojenými CAN sběrnici jsou data přenášeny pomocí zpráv (rámců). Tyto zprávy se mohou týkat přenosu dat (datová zpráva nebo žádost o data) a nebo mohou řídit komunikaci na sběrnici (chybová zpráva a zpráva o přetížení).

2.4.1 Datová zpráva (Data Frame)

Tato zpráva (viz. obr 2.2) obsahuje data (*Data Field*), která mají být přenesena mezi zařízeními a další kontrolní nebo přenosové informace (*SOF*, *Arbitration Field*, *Control Field*, *CRC*, *ACK* a *EOF*).

- Datová zpráva začíná bitem *SOF (Start of Frame)*, který značí začátek zprávy. Je určen jedním časovým segmentem dominantní úrovně. SOF se používá kromě u datových zpráv také u požadavků na data (Remote Frame, viz. 2.4.2). *SOF* je základním způsobem synchronizace přenosu zpráv.
- *Arbitration Field* obsahuje identifikátor a *RTR-BIT*. Identifikátor může být klasický a obsahovat 11 bitů a nebo může být rozšířený na 29 bitů. Rozšířený identifikátor obsahuje doplňkové informace pro upřesnění rozlišení zprávy. Dále toto pole obsahuje RTR-BIT (Remote Transission Request BIT) pro rozlišení jedná-li se o datovou zprávu a nebo o požadavek na data.
- *Control Field* nese informaci o počtu datových bytů ve zprávě.
- *Data Field* obsahuje až 8 bytů pro přenášená data.
- *CRC* a *ACK* jsou zabezpečením používaným při přenosu zpráv, které je popsáno v předchozí části (viz. 2.3).

Konec přenosu zprávy je dán polem EOF, které je dáno 7-mi recesivními bity. Po aktuálním přenosu, může začít nový přenos až po vyslání pole dalších 3 recesivních bitů.



Obrázek 2.2: Datová CAN zpráva

2.4.2 Žádost o data (Remote Frame)

Tento typ zprávy se liší od datové zprávy pouze v nastaveném *RTC-BIT*u a v přítomnosti datové části zprávy. Používá se jako žádost o určitá data. Ve zprávě žádosti o data se nastaví identifikátor odpovídající zprávě, kterou chceme získat.

2.4.3 Chybová zpráva (Error Frame)

Chybové zprávy slouží k okamžité signalizaci chyb na sběrnici. Každá odlišnost od CAN standardu je okamžitě detekována a pomocí chybovým zpráv je na ni upozorněno. Chybová zpráva se skládá z chybového příznaku (Error Flag) a oddělovače (Error Delimiter).

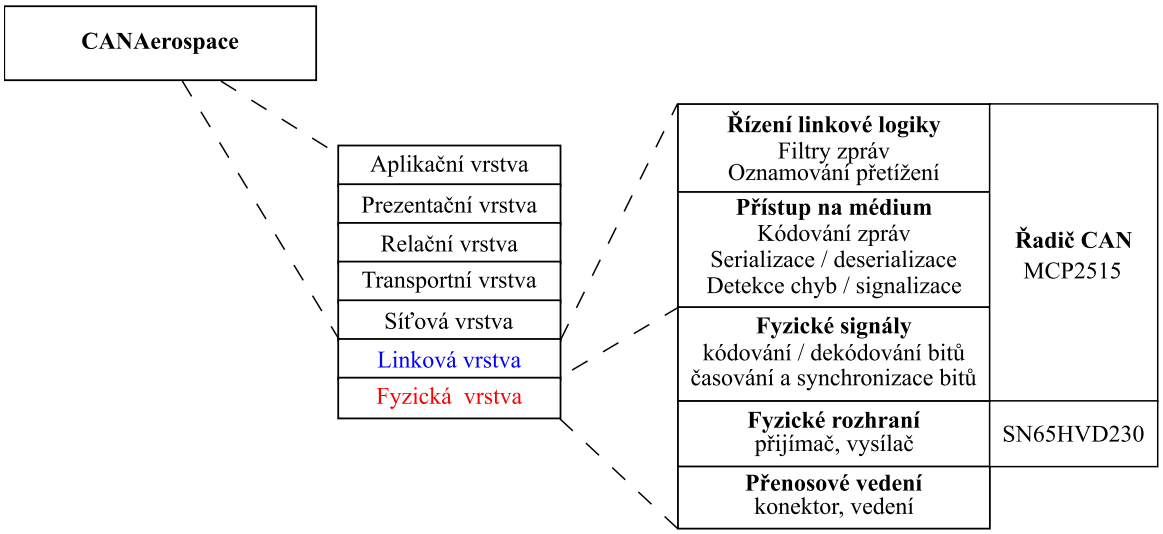
Chybový příznak se skládá z 6 stejných bitů, což porušuje pravidlo Bit Stuffing (viz. 2.3). Oddělovač ve zprávě obsahuje 8 recesivních bitů a dává prostor pro vložení chybových příznaků ostatními zařízeními na sběrnici.

2.4.4 Zpráva o přetížení (Overload Frame)

Tato zpráva umožňuje pozastavení a zpoždění datové nebo řídicí komunikace na sběrnici. Většinou je vysílána zařízením, které je přetíženo komunikací.

2.5 Architektura Controller Area Network

Standardní komunikační protokoly jsou abstrahovány pomocí sedmivrstvého referenčního modelu ISO/OSI (viz. obr 2.3). Protokol CAN implementuje pouze základní fyzickou a linkovou vrstvu. Zbylé komunikační vrstvy mohou vytvořeny programově za použití některého z vyšších protokolu CAN. Implementovaný řadič sběrnice CAN bude využívat pro svou komunikaci protokol CANAerospace [14]. Což je jeden ze standardu HLP (Higher Layer Protocol) pro CAN. CANAerospace byl navrhnut pro vysoce spolehlivou komunikaci mezi systémy na bázi mikropočítačů v aplikacích využívaných v leteckém průmyslu. Cílem CANAerospace protokolu je definice komunikačního prostředku pro aplikace vyžadující efektivní monitorování dat a jednoduchou synchronizaci posílaných rámců bez nadbytečného zatížení systému.



Obrázek 2.3: Referenční model ISO/OSI

2.6 Protokol CANAerospace

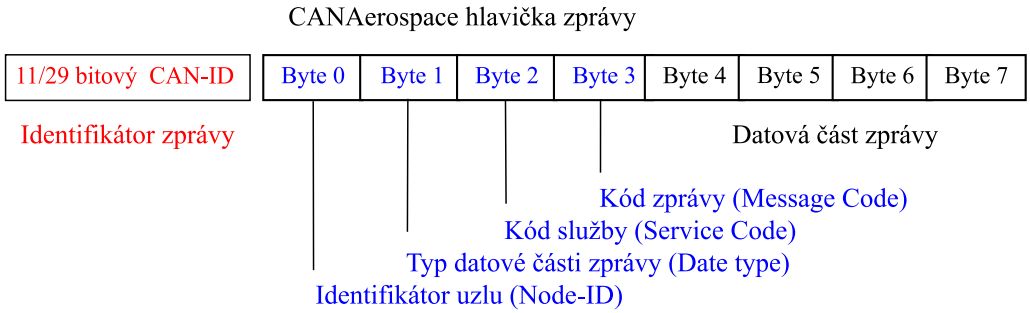
CANAerospace je otevřený standard velmi jednoduchého a nenáročného protokolu, který byl původně vyvinut pro použití v letecké technice. CANAerospace byl navržen pro vysoce spolehlivou komunikaci mikropočítačových systému v leteckých aplikacích.

Definuje 6 základních typů zpráv, které jsou určeny pro různé druhy služeb, jako je např. typ EED (Emergency Event Data) pro službu vyžadující okamžitou akci nebo služba IDS

pro identifikaci zařízení. Pomocí těchto zpráv budou komunikovat dva účastníci, využívající zařízení s CAN sběrnicí jako prostředníka. Každá z těchto služeb je spojena s CAN-ID definujícím prioritu zprávy. Pro reprezentaci dat ve zprávě protokol CANAerospace podporuje 8 a 16 bitové základní datové typy. Byty jsou v CAN zprávách uspořádány pomoci kódování typu „Big Endian“. Všechny CAN zprávy obsahují 4 bytovou hlavičku pro identifikaci a 1 až 4 byty pro datovou část zprávy.

2.6.1 Formát CAN zprávy

Standardní formát CAN zprávy (viz. obr 2.4) používá 4 byty pro hlavičku identifikující komunikační uzel, typ datové části, kód zprávy a kód služby.



Obrázek 2.4: Standardní CAN zpráva definována protokolem CANAerospace

Kapitola 3

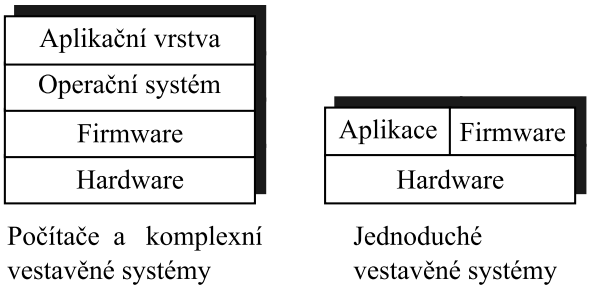
Vestavěné systémy

Vestavěné (Embedded) systémy jsou speciální samostatná zařízení, obsahující hardwarovou i programovou část . Většinou jsou vytvořeny za účelem plnění jediného úkolu. Mohou být součástí větších systému, kterým poskytují požadovanou inteligenci. Jsou konfigurovány pro optimální výkon a malou spotřebu.

Počítačové a vestavěné systémy obsahují funkcionální vrstvy [9], které spolu úzce souvisí a společně komunikují (viz. obr 3.2). Na nejnižší úrovni je vždy hardware, ten je konfigurován a řízen pomocí programu spuštěném procesorem ihned po zapnutí systému. Toto základní programové vybavení se nazývá *firmware*. Nad firmwarem bývá u malých a jednoduchých vestavěných zařízení ihned aplikační vrstva, u komplexnějších systémů je navíc nad firmwarem umístěn operační systém.

Ve větších vestavěných zařízeních (počítačích) existuje ve firmware speciální program *bootloader*, který po spuštění procesorem načte operační systém z disku do operační paměti a spustí ho. Operační systém ovládá počítač, spouští programové aplikace a umožňuje uživateli základní přístup k hardwarovému vybavení systému.

V běžícím operačním systému mohou být spuštěny různé aplikace komunikující s uživatelem a používající hardware vestavěného systému.



Obrázek 3.1: Vestavěné systémy

Pro oživení vestavěného systému lze použít mnoho variant dostupných operačních systému[1]. U vestavěných OS je kladen důraz na jejich minimální velikost, rychlost, modulárnost, rozšířitelnost a snadnou přenositelnost na různé hardwarové platformy. Některé

[1]Seznam embedded OS lze najít na adrese http://en.wikipedia.org/wiki/List_of_operating_systems#Embedded

systémy jsou speciálně navrhnuty pro použití v *real-time* systémech (RTOS), které vyžadují vysokou schopnost práce v reálném čase.

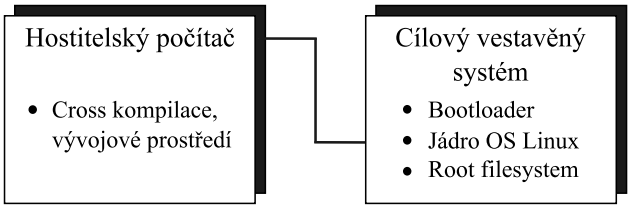
3.1 Embedded Linux

Embedded Linux [16] je navrhnut pro vestavěné systémy využívající Linuxové jádro. Vestavěné zařízení může obsahovat oficiálně vydanou verzí jádra nebo verzi, která je speciálně upravená pro specifické použití na dané hardwarové platformě. Tato modifikovaná verze může být dodatečně optimalizována pro určitý hardware, může obsahovat nadstandardní vývojové a ladící nástroje a nebo může být přidána podpora pro nový, předchozí verzí nepodporovaný, hardware.

Aby mohl být ve vestavěném systému umístěn OS Linux, musí toto zařízení disponovat nejméně 32-bit procesorem, jednotkou MMU (viz. 4.1.2) a dostatečnou kapacitou operační paměti RAM pro bezproblémový běh systému.

Ve vestavěných systémech, které neobsahují MMU, lze použít OS *uClinux*. V systémech, které MMU jednotku mají, jsou všechny procesy a data mapovány do virtuální paměti a MMU jim přiděluje adresy ve fyzické paměti.

V systému bez MMU a tudíž bez virtuální paměti je každému procesu přiděleno místo uvnitř fyzické paměti, z kterého může být spuštěn. Přidělená oblast paměti je pevná, uClinux ji nemůže rozšířit, protože ostatní procesy mohou být umístěny ve vedlejších oblastech.



Obrázek 3.2: Vývoj pro cílový vestavěný systém

3.2 Vytvoření cílového systému

Cílový vestavěný Linux [16] je vytvořen pomocí konfigurace, kompilace a sestavení všech potřebných komponent. Proces sestavení cílového systému lze popsat těmito kroky:

- Vymezení použitých komponent v systému.
- Konfigurace a sestavení linuxového jádra. Jádro systému pro vestavěné systémy obsahuje pouze knihovny a aplikace, potřebné k spuštění a chodu vestavěného systému.
- Vytvoření kořenového souborového systému pro použitou platformu (viz. 4.4.1).
- Nastavení a spuštění zavaděče (viz. 4.4.2).

Kapitola 4

Vývojový kit Atmel ATNGW100

4.1 Architektura AVR32

AVR32 [6] je nová moderní mikroprocesorová architektura vyvinutá společností Atmel. Na které je založeno vysoce výkoné 32 bitové RISC jádro mikroprocesorů, navrhnutích pro vestavěné systémy a malou spotřebu energie. Instrukční sada architektury zavádí různorodé mikroarchitektury, které dovolují AVR32 procesorům pracovat jako nízko, středně a vysoce výkonné.

AVR32 implementuje AVR32A a AVR32B mikroarchitekturu. Typ AVR32A je určen pro cenově závislé zařízení a koncové aplikace, jako jsou menší mikroprocesory. Tato mikroarchitektura nenabízí speciální registry pro přerušení a neobsahuje registry pro uložení návratové adresy a status registrů, ale všechny tyto informace jsou automaticky ukládány do zásobníku. Řadič přerušení může volně využívat všech registrů do ukončení obsluhy přerušení, poté jsou všechny informace obnoveny ze zásobníku. Zásobník je také určen pro ukládaní status registru a návratových adres dojde-li k výjimce nebo systémovému volaní.

Typ architektury AVR32B je cílen pro aplikace, u kterých je kladen důraz na dobu zpracování přerušení, jsou zde také implementovány speciální registry pro ukládání hodnot status registru a návratových adres pro přerušení, výjimky a systémová volání. Zpoždění, způsobené prací se zásobníkem je zde odstraněno.

4.1.1 Režimy procesoru

AVR32 procesor se může nacházet v několika různých stavech vykonávaní. Základním režimem je stav *RISC* (RISC State), který definuje několik módu vykonávaní, dělených podle priorit a přednostních úrovní (viz. tab 4.1). Tyto módy mohou být změněny programově a nebo vlivem externího přerušení či vznikem výjimky. Aktuální vykonávaní v nastaveném módu může být přerušeno potřebou přepnutí do módu s vyšší prioritou. Při spuštění AVR32 procesoru je nastaven aplikační mód. V tomto módu nejsou dostupné některé systémové registry a chráněné oblasti paměti, na rozdíl od ostatních „systémových“ módů.

Dalším režimem, v kterém se může AVR32 procesor nacházet je režim ladění (*Debug State*). V tomto režimu jsou implicitně vypnuta všechna přerušení a je povolen přístup k chráněné paměti všem registrům, což podporuje implementaci ladicích rutin s možností plné kontroly systému.

Posledním stavem, který některé AVR32 procesory podporují je stav pro zpracování Javy (*Java State*). Hardwarově nenáročné vykonávání Java byte kódu je implementováno

Priorita	Mody vykonávaní	Úroveň	Popis
1	Nemaskovatelná přerušení	Přednostní	Nejvyšší priorita zpracovaní
2	Výjimky	Přednostní	Zpracovaní výjimek
3 - 6	Přerušení	Přednostní	Priorita zpracování je dána hlavním účelem použití
-	Systémový režim	Přednostní	Spuštěn po restartu procesoru
-	Aplikační režim	N	Normální vykonávaní programu

Tabulka 4.1: Základní stavy vykonávaní v RISC režimu procesoru

zahrnutím přídavného Java Extension modulu (JEM) a softwarové podpory JVM (Java Virtual Machine), obsahující doplňkovou instrukční sadu pro Javu.

4.1.2 Paměťová jednotka MMU

AVR32 architektura definuje volitelnou paměťovou jednotku MMU (Memory Management Unit), která umožňuje efektivně implementovat virtuální paměť a spravovat velké paměťové prostory. Dále dovoluje pracovat s různými sekcemi paměti a zjednodušuje více procesní vykonávání

MMU je zodpovědná za namapování 32-bitového prostoru virtuální paměti do 32-bitové fyzické oblasti paměti. Pro namapovaní virtuálního adresového prostoru do fyzického prostoru se používá princip stránkování. Stránkované bloky paměti mohou mít velikost 1, 4 a 64 KB, podporována je i velikost 1 MB. Každá stránka obsahuje tabulku s informacemi o obsažených datech, které jsou potřebné při překladu adres.

AVR32 procesory používají pro zrychlení překladu virtuálních adres na fyzické speciální vyrovnávací paměť TLB (Translation Lookaside Buffer), která obsahuje nastavitelný počet nejčastěji používaných stránek. Vedle TLB jsou v procesoru také umístěny

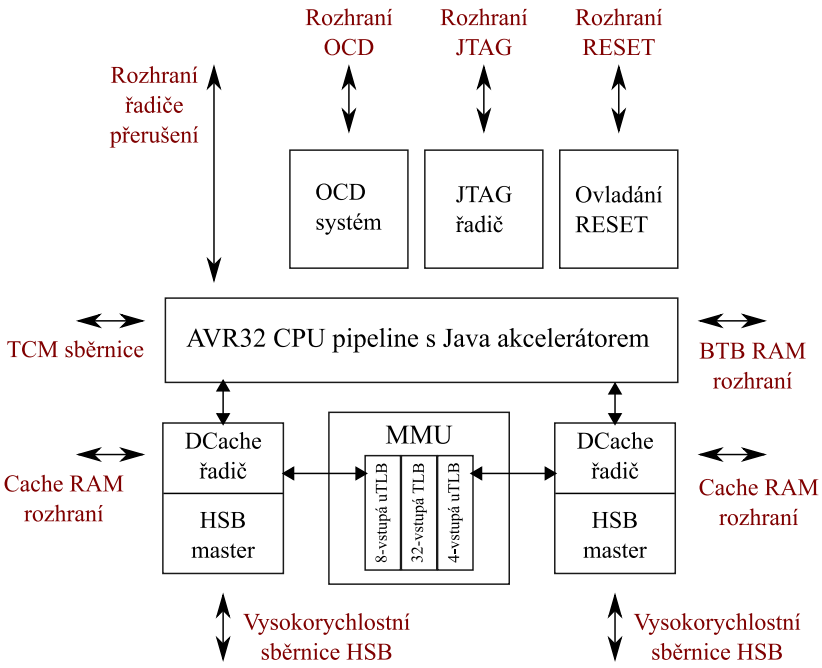
4.1.3 Ochrana paměti pomocí MPU

Kromě jednotky MMU zahrnuje architektura AVR32 jednotku ochrany paměti MPU (Memory Protection Unit), což je zjednodušená MMU, která ale neprovádí žádný překlad adres. Jednotka MPU dovoluje rozdělit paměťový prostor do různě chráněných oblastí, kdy je uživatelsky specifikována velikost chráněného prostoru a jeho začátek na určité adrese v paměti. MPU podporuje bloky paměti o velikosti od 4 KB do 4 GB s nutnosti zarovnání adres dat na začátek chráněné oblasti.

Každé z chráněných oblastí jsou přiděleny přístupová práva. MPU při přesunu paměti kontroluje přístupová práva operace a pokud se neshodují s právy paměťového bloku vzniká výjimka a operace je zrušena.

4.2 AVR32 Aplikační procesor

Vývojový kit Atmel ATNGW100 je řízen kompletním „System-on-chip“ AVR32 [5] aplikačním procesorem AP7000 (viz. obr B.2) s RISC jádrem bežícím na frekvenci 150Mhz. Procesor AP7000 (viz. obr 4.1) obsahuje MMU jednotku pro řízení paměti a implementuje řadič přerušení podporující moderní operační systémy a systémy schopné práce v reálném čase. Procesor AP7000 je založen na architektuře AVR32B.



Obrázek 4.1: Blokové schéma AVR32 AP procesoru

Tento AVR32 procesor obsahuje DSP (Digital Signal Procesor) instrukční sadu, určenou a optimalizovanou pro numerické zpracování dat, a SIMD (Single Instruction Multiple Data) instrukční sadu, navrženou pro účinné paralelní zpracování dat.

4.2.1 Pipeline

Využívá sedmistupňového zřetězení zpracování instrukcí pipeline. Pipeline obsahuje další tři podjednotky (ALU pipeline, Multiply pipeline a Load-Store pipeline), které mohou vykonávat od sebe různé instrukce paralelně. Instrukce mohou být zpracovány v pořadí v jakém přišli a nebo mimo pořadí. A díky tomu mohou být aritmetické operace použity na jakákoliv nezávislá data.

- ALU pipeline slouží pro zpracování instrukcí manipulujícími s daty, jako jsou aritmetické a logické operace.
- Multiply pipeline zpracovává všechny násobící instrukce.
- Load-Store pipeline je schopná číst a nebo zapisovat instrukci do dvou registrů během jednoho hodinového cyklu.

4.2.2 Paměti

Procesor má zabudovanou SRAM paměť s rychlým a bezpečným přístupem o velikosti 32 KB, která je tvořena pomocí dvou 16 KB bloků.

Systémová sběrnice je implementována jako HSB (High Speed Bus) matice, kde jsou všechny adresy pevně dané. Matice sběrnice má několik master a slave zařízení. Každé master zařízení má vlastní sběrnici a vlastní dekodér. To dovoluje použít odlišné způsoby mapování pro každé z master zařízení, na která jsou připojena slave zařízení.

4.3 Periferie

4.3.1 PIO piny

Na vývojové desce je umístěno 32 programovatelných PIO (Paraller Input/Output Controller) piny. Tyto piny lze jednoduše ovládat pomocí nastavovatelných registrů.

Každý z I/O pinů má určen hlavní a vedlejší účel. U všech pinů lze detekovat přerušení při změně úrovně napětí. U těchto pinů lze také nastavit jejich viditelnost, schopnost chovat se jako výstup, dále je možné nastavit každému pinu vstupní *glitch* filtr, který zachytává zákmity a rušivé signály.

PIO piny jsou multiplexované a každému je přiřazena jedna nebo dvě funkce spojená s vestavěnými periferiemi. Některé piny jsou ale nevyužité.

Na nevyužité piny je, v případě našeho implementovaného vestavěného systému, připojen rozšiřující hardware CAN sběrnice a RTC modulu (viz. schéma zapojení **??**)

4.3.2 SPI rozhraní

SPI (Serial Peripheral Interface) rozhraní tvoří sériový datový kanál pro komunikaci s externími zařízení v *master* nebo *slave* módu. Toto rozhraní je implementováno jako posuvný registr, který vysílá sériová data ostatním SPI rozhraní, připojeného hardware.

Při komunikaci je SPI rozhraní, ovládající datový tok, v režimu master a rozhraní přijímající nebo odesílající data v režimu slave.

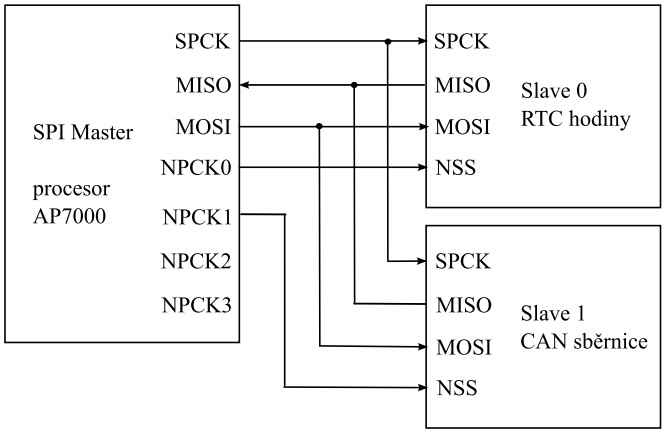
SPI rozhraní se skládá ze dvojice datových a dvojice řídících kanálu:

- MOSI (Master Out Sleve In) - Je datový kanál přesouvající data z master zařízení na vstup slave zařízení.
- MISO (Master In Slave Out) - Je datový kanál přesouvající data z slave zařízení na vstup master zařízení.
- SPCK (Serial Clock) - Tento kanál je řízen master zařízením a dovoluje regulovat datový tok. Master rozhraní tak může měnit rychlost datového toku.
- NSS (Slave Select) - Tento kanál umožňuje ovládání a volbu slave zařízení.

Pro přenos dat přes SPI rozhraní je nutné nastavit stejný způsob synchronizace hodinového signálu. Fázi a polaritu hodinových impulsů lze nastavit odpovídajícími registry procesoru a každé konfiguraci odpovídá jeden ze čtyř SPI modu. b

4.3.3 Ostatní hardware

Procesor dále obsahuje rozhraní externí sběrnice EBI (External Bus Interface), které přenáší data mezi externí pamětí nebo periferních zařízení a systémovou HSB sběrnicí. EBI rozhraní integruje řadiče pro SDRAM, DataFlash, SRAM, multimediální MMC a SD karty. V zařízení je vestavěna 32 MB SDRAM paměť, sériová NAND a paralelní paměť NOR o velikostech 8 MB. Dále je na desce vestavěn ladící systém Nexus s rozhraním JTAG, dva ethernetové porty s rychlostí 10/100 Mbps, klasický sériový port, univerzální sériové vysokorychlostní rozhraní USB 2.0, LCD řadič s rozhraním pro připojení TFT displejů, podporující maximální rozlišení 2048 x 2048, AC97 zvukový systém, RTC modul s 32 bitovým čítačem, WDT časovač a jiné.



Obrázek 4.2: SPI komunikace master a slave

4.4 Vestavěný operační systém Linux

4.4.1 Souborový systém

Při portování OS Linux do vestavěného systému nastává problém [8], jak vytvořit souborový systém bez klasického uložení dat na pevném disku. Ve vestavěných zařízeních se používají různé typy flash paměti, které se liší použitou technologii ve zpracování a uložení dat.

Obvykle používané souborové systémy, jako je např. standardní systém ext2, nelze ve flash pamětích účinně použít, protože se liší velikostí používaných paměťových bloků. Dále jsou některé paměti omezeny počtem zápisových operací. Proto je velmi žádoucí použít ve vestavěném systému některý ze speciálních souborových systému s vysokou kompresí dat.

Souborové systémy navržené pro uchování dat v flash pamětích se liší způsobem použití v jádře OS Linux a mechanismy uložení dat (viz. obr 4.3). Ve vestavěných systémech lze použít tyto základní souborové systémy:

- *initrd* souborový systém byl vyvinut pro malé linuxové systémy. Při zavedení operačního systému nejdříve zavaděč zkopíruje obraz souborového systému initrd z flash paměti do RAM, rozbalí se jádro a začne proces inicializace. Poté se připojí souborový systém, jehož obraz se rozbalí do operační paměti.

- *cramfs* je komprimovaný read-only souborový systém. Je zahrnut v současných jádrech systému Linux.

- *ramfs* je souborový systém, který je uložen v paměti RAM. Často se používá ve souborových systémech pro flash paměti k uchování dočasných a často měněných dat.

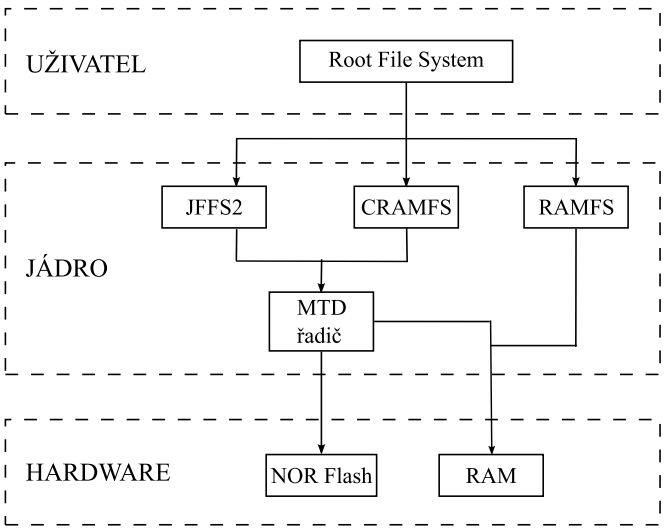
- *jffs2* souborový systém je navrhnut pro použití ve flash pamětích. Dovoluje kompresi dat a přístup pro čtení i zápis. jffs2 nemaže uložená data, pouze je přepisuje novějšími. V případě že dojde k zaplnění kapacity paměti, spustí se garbage collector, který odstraní veškerá nadbytečná data.

Ve vestavěném systému je použit souborový systém jffs2. Tento systém lze pro platformu AVR32 procesoru vytvořit pomocí programu *Buildroot*[1].

[1]Informace o programu Buildroot lze najít na adrese http://www.atmel.no/buildroot/buildroot.html

Buildroot je skupina skriptů, pomocí kterých lze pro danou platformu vytvořit celý souborový systém. Jeho použití je následující:

- Nejdříve je nutné nastavit konfiguraci pro vývojový kit Atmel ATNGW100 pomocí příkazu `make atngw100_defconfig`
- Pro pokročilou a námi definovanou konfiguraci lze použít příkaz `make menuconfig`. Pomocí toho příkazu spustíme grafickou konfigurační utilitu menuconfig, díky které lze provést mnohem obsáhlejší konfiguraci.
- Proces vytvoření souborového systému spustí příkaz `make`



Obrázek 4.3: Souborový systém ve vestavěných systémech [8]

4.4.2 Zavaděč systému

Na vývojové desce je nainstalován zavaděč U-Boot[2]. U-Boot slouží primárně pro zavedení operačního systému. Typicky je U-Boot umístěn ve vestavěných systémech v paralelní paměti typu NOR.

U-Boot je navržen pro zavedení a start systému v maximální rychlosti. Hned, jak je to možné, jsou zapnuty vyrovnávací paměti procesoru a během průběhu zavádění systému jsou inicializována pouze používaná zařízení.

Zavaděč U-Boot podporuje různé způsoby zavedení systému. Umožňuje automatické zavedení podle uložených parametrů po uplynutí časové prodlevy. Tento zavaděč podporuje kromě zavedení operačního systému také možnost nahrání programového firmware do paměti.

Upgrade zavaděče na vývojovém kitu Atmel ATNGW100 je jednoduše proveditelný pomocí programovatelného rozhraní JTAG, přes které se lze na desku připojit mimo standardních JTAG programátorů také pomocí klasického paralelního portu.

[2]Informace o systémovém zavaděči U-Boot lze najít na adrese http://www.denx.de/wiki/U-Boot

4.4.3 Úprava jádra

Ve vyvíjeném vestavěném systému je použito upravené linuxové jádro verze 2.6.24.3.atmel.3[3]. Do tohoto jádra byly dodatečně přidány některé nadstandardní ovladače a byly na něj aplikovány opravné záplaty.

Pro použití ve vyvíjeném vestavěném systému je nutné provést úpravu zdrojového kódu jádra. Je nutné přidat podporu pro nové SPI zařízení.

V souboru `linux-2.6.24.3.atmel.3/arch/avr32/boards/atngw100/setup.c` je nutné upravit definici struktury `spi_board_info`. Tato struktura obsahuje informace o stávajících SPI zařízeních, které definují parametr `modalias` pro nastavení asociace s určitým ovladačem, maximální přenosovou rychlost `max_speed_hz` a nastavení chip selectu `chip_select`.

Ve struktuře `spi0_board_info` jsou informace o SPI zařízení používaném při komunikaci datové flash paměti.

Přidáme informace o nových SPI zařízeních pro CAN sběrnici a RTC modul. Upravená struktura bude vypadat následovně:

```
static struct spi_board_info spi1_board_info[] __initdata = {
   {
       .modalias       = 'spidev',
       .max_speed_hz   = 8000000,
       .chip_select    = 0,
   },{
       .modalias       = 'spidev',
       .max_speed_hz   = 8000000,
       .chip_select    = 1,
   },
};
```

Poté je nutné upravit funkci `atngw100_init()`, která provádí inicializaci zařízení.

```
static int __init atngw100_init(void)
{
   ...
   at32_add_device_spi(0, spi0_board_info, ARRAY_SIZE(spi0_board_info));

   /* přidání dalšího SPI zařízení */
   at32_add_device_spi(1, spi1_board_info, ARRAY_SIZE(spi1_board_info));
   ...
}
```

Kompilace jádra

Kompilaci upraveného jádra provedeme pomocí příkazů:

```
make ARCH=avr32 CROSS_COMPILE=avr32-linux- atngw100_defconfig
make ARCH=avr32 CROSS_COMPILE=avr32-linux-
make ARCH=avr32 CROSS_COMPILE=avr32-linux- modules_install
```

[3] Jádro 2.6.24.3.atmel.3 a další lze najít na adrese http://avr32linux.org/twiki/bin/view/Main/LinuxPatches#2_6_24_3_atmel_3

Kapitola 5

Hardwarové řešení embedded systému

5.1 Technický návrh zařízení

Vestavěný systém je implementován na vývojové desce Atmel ATNGW100. Tato deska disponuje aplikačním AVR32 procesorem AP7000, který řídí veškerou vnitřní i vnější komunikaci hardwarového vybavení v systému. Na vývojové desce jsou vyvedeny PIO piny, na které je připojena externí deska, obsahující CAN sběrnici a modul RTC.

Na nevyužité PIO piny jsou vyvedena SPI rozhraní jednotlivých komponent externí desky. SPI zařízením v režimu master je vždy AVR32 procesor, který řídí komunikaci s ostatními slave zařízeními.

5.2 Fyzická implementace CAN sběrnice

V systému kde bude k vývojovému kitu ATNGW100 připojena externí deska s CAN sběrnicí jsem použil pro řízení sběrnice kontroler MCP2515 [12] od firmy Microchip a pro implementaci fyzické vrstvy sběrnice je použito zařízení SN65HVD230 [15] od firmy Texas Instruments. CAN sběrnice je připojena k desce ATNGW100 přes GPIO porty. Procesor na desce přes ně komunikuje pomocí rozhraní SPI. Vstupy a výstupy z CAN kontroleru jsou propojeny s odpovídajícími vstupy a výstupy na CAN přijímači. Ten je připojen ke klasickému CAN konektoru CANNON.

5.2.1 CAN kontroler MCP2515

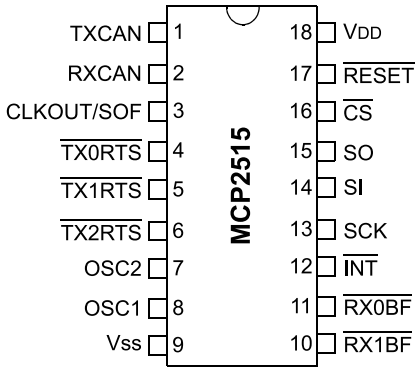
CAN sběrnice je řízena pomocí samostatného kontroleru MCP2515 (viz. obr 5.1, [12]). Tento řadič používá CAN specifikaci verze 2.0B. Je schopen přijímat všechny typy zpráv (viz. 2.4) a pro filtraci CAN zpráv obsahuje 2 akceptační masky a 6 filtrů.

Tento CAN kontroler je ovládán procesorem, s kterým probíhá komunikace pomocí SPI rozhraní. MCP2515 může operovat v pěti různých funkčních módech. Mimo módu v kterém probíhá normální komunikace, může tento řadič pracovat v konfiguračním módu, sleep módu, naslouchacím (listen-only) módu a v loopback režimu.

Do konfiguračního režimu je řadič přepnut ihned po zapnutí, zde je možno nastavit všechny kontrolní registry, filtry a masky.

Pro účel testování, kdy není dostupná fyzická CAN sběrnice, je možné použít *loopback* režim.

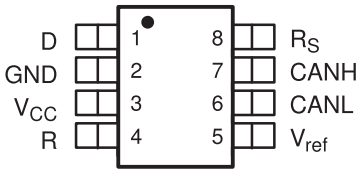
Jak již bylo zmíněno výše, CAN řadič je řízen pomocí svého SPI rozhraní, pro které jsou implementovány speciální instrukce se specifickou funkcí a využitím.



Obrázek 5.1: CAN kontroler MCP2515 [12]

5.2.2 CAN Přijímač SN65HVD230

Přijímač SN65HVD230 (viz. obr 5.2, [15]) realizuje rozhraní na fyzické vrstvě mezi CAN sběrnicí a kontrolerem MCP2515. V navrženém vestavěném systému je použit, protože realizuje operace na napětí 3.3V, které používá také vývojový kit Atmel ATNGW100. Tento obvod je řízen pouze kontrolerem MCP2515.



Obrázek 5.2: CAN přijímač SN65HVD230 [15]

SN65HVD230 podporuje tři základní funkční módy. Přijímač může pracovat ve vysoko rychlostním módu (high-speed), v slope-control módu a v režimu s minimální spotřebou energie (low-power). Výběr těchto režimu se ovládá pomocí pinu Rs, na které je nutné přivést určitou napěťovou úroveň.

Vysokorychlostní režim dovoluje transfer dat bez nějakých větších omezení. V tomto případě je rychlost přenosu omezena pouze délkou datového vodiče.

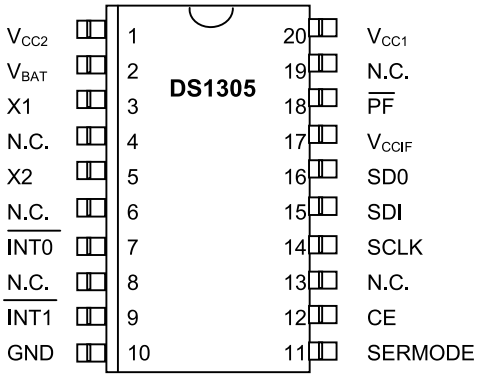
Slope-control režim je určen pro použití v aplikacích, které používají nestíněný kabel pro snížení ceny výsledného zařízení. Hlavním přínosem toho režimu je snížení elektromagnetického vyzařování.

5.3 Modul RTC hodin

RTC modul DS1305 (viz. obr 5.3, [11]) jsou sériové hodiny, které navíc poskytují budík a kalendář, implementovaný pomocí BCD kódování. Modul DS1305 podporuje sériovou

komunikaci a také přenos dat a adres pomocí SPI rozhraní. Toto rozhraní je propojeno s odpovídajícími vstupy a výstupy na vývojovém kitu, s jehož vestavěným procesorem komunikuje.

DS1305 implementuje klasický způsob čtení a zápisu po jednom byte. Při kontinuálním zápisu dat do adresy registru, se tato adresa paměti automaticky inkrementuje. Byte se specifickou adresou registru je vždy prvním zapsaným bytem po signálu CE (Chip enable). Po odeslání adresy mohou být přečteny nebo zapsány data. Tento RTC obvod také podporuje tzv. burst mod, což je mód mnohonásobného čtení nebo zápisu s automatickou inkrementací cílové adresy.



Obrázek 5.3: RTC hodiny [11]

Kapitola 6

Jádro operačního systému

6.1 User space a kernel space

V operačním systému Linux se můžeme nacházet ve dvou specifických režimech. Tím prvním je uživatelský režim, ve kterém běží uživatelské aplikace. V tomto režimu je možné používat pouze omezenou instrukční sadu a přistupovat, do pouze pro uživatele vyhrazené, části paměti, zvané *user space*. Druhým režimem je privilegovaný režim jádra, v kterém je možné přistupovat do celé oblasti paměti, zvané kernel space, a používat plnou instrukční sadu.

6.2 Modulární přístup

Jádro OS Linux implementuje modulární přístup [10]. Který je ale volitelný, jádro lze zkompilovat i bez podpory modulů.

Jádro je rozděleno na základní celek a na moduly, což mohou být ovladače nebo části, přinášející jádru novou funkčnost. Tato modularita se vyznačuje jednotným rozhraním, kdy všechny moduly mohou používat exportované symboly z hlavičkových souborů jádra. Přidané moduly v jádře se mohou stát jeho pevnou součástí, což závisí na způsobu slinkování se stávajícím jádrem. Moduly se do jádra mohou zavádět automaticky při startu systému, to závisí na základě způsobu jeho použití.

6.3 Souborové operace

V Linuxu platí, že téměř vše je soubor. Kromě normálních souborů existují také speciální soubory jako jsou symbolické odkazy, roury, sockety, fronty a soubory zařízení. Pro implementaci řadiče v jádru budou velmi důležité soubory zařízení. Které mohou být vytvořeny při vložení modulu řadiče do jádra. Soubory zařízení můžeme používat pomocí souborových operací [10]. Každý soubor, který je v rámci určitého procesu používán je identifikován svým deskriptorem.

Aby bylo možné používat soubory zařízení a obecně soubory, je nutné je nejdříve otevřít. Otevření znamená, že jádro nejdříve vytvoří a naalokuje datové struktury, potřebné pro přístup do souboru, soubor označí jako otevřený daným procesem a sám proces získá deskriptor, který na otevřený soubor odkazuje. Otevření souboru implementuje funkce `open()`, která jako parametry očekává cestu k souboru a konstantu, určující režim přístupu k souboru.

Pokud otevřený soubor už nepotřebujeme, je nutné ho zavřít. Neuzavřeme-li soubor, jádro jej zavře automaticky ve chvíli ukončení procesu, kterým byl otevřen. Zavřením nepoužívaných souboru se odstraní alokované prostředky v paměti. Pro uzavření souboru slouží funkce `close()`, která očekává jako vstupní parametr pouze platným deskriptor souboru.

S otevřeným souborem se nejčastěji provádí operace čtení a zápisu, implementované funkcemi `read()` a `write()`. Tyto operace pracují s částí paměti, do které se načítají data a nebo ze které se zapisují. Během těchto operací se mění aktuální pozice v souboru. Je-li soubor otevřen pro čtení nebo zápis, je jeho aktuální pozice nastavena na začátek. V průběhu souborových operací se tato pozice mění a odráží tak skutečnost o již zpracovaných datech.

Se soubory lze provádět mnoho dalších operací. Existují např. řídící operace implementované funkcemi `fcntl()` a `ioctl`, které dovolují měnit režim práce se souborem nebo zamknout ho a tím zamezit přístupu k němu od jiných procesů. Dále jsou dostupné operace pro asynchronní přístup, pro pohyb v souboru a rozšířené základní operace.

Kapitola 7

Analýza prostředků a návrh implementace řadiče sběrnice CAN

7.1 Cross kompilace

Proces, kdy pro vytvoření spustitelného programu použijeme kompilátor na jiné platformě, než pro kterou je určen se nazývá Cross kompilace. Tento způsob kompilace zdrojového kódu, který bude spustitelný na cílovém počítači nebo vestavěném systému, striktně závisí na architektuře použitého procesoru.

Jádro OS Linux i řadič CAN sběrnice, stejně tak jako všechny ostatní kompilovatelné zdrojové kódy programů, musí být pro použití na vyvíjeném vestavěném systému překládány pro cílovou architekturu AVR32.

7.2 Návrh řadiče

Cílem je, aby programový ovladač řadiče zpřístupnil hardware uživatelským procesům a dokázal zprostředkovat případnou vzájemnou komunikaci.

Programový řadič, po vložení do jádra, inicializuje veškerý hardware využívaný ve vestavěném systému. Tato inicializace bude záviset na tom, zda byly při vložení modulu řadiče do jádra zadány některé z implementovaných parametrů. Při vložení modulu do jádra, bude možné specifikovat přenosovou rychlost a komunikační režim CAN sběrnice, dále bude možné nastavit aktuální čas a datum v RTC modulu.

Pro inicializaci kontroleru MCP2515 (viz. 5.2.1) je nutné, aby se kontroler nacházel v konfiguračním módu. Poté se nastaví odpovídající registry. Nastavení i další operace s MCP2515 probíhají pomocí jeho SPI rozhraní, které definuje funkce pro implementaci základních operací s registry CAN zařízení. Tyto definované základní operace budou implementovány pomocí primitiv pro čtení a zápis jednoho byte na SPI rozhraní.

Proces inicializace RTC modulu (viz. 5.3) zajišťuje pouze start oscilátoru, který se provede pomocí zápisu byte do registru,

Poté, co budou tato zařízení inicializována, provede se zaregistrování obslužné rutiny přerušení, které vznikne při přijetí zprávy po CAN sběrnici.

Po úspěšné inicializaci všech zařízení, modul vytvoří v adresáři `/dev` virtuální zařízení a zaregistruje jejich souborové operace. Tato virtuální zařízení jsou soubory, z kterých bude možné číst data a také je zapisovat. Pokud bude soubor zařízení čten, budou se číst i data, která vlastní, analogicky je tomu tak i v případě zápisu dat.

V případě čtení souboru zařízení RTC modulu, bude RTC požádáno o zjištění aktuálního data a času a tyto hodnoty budou vráceny uživatelskému procesu. Dojde-li k zápisu informací o datu a čase, přenastaví se RTC na tyto nové hodnoty. Při čtení souboru zařízení CAN, budou přečtenými daty aktuálně přijaté zprávy, které byly přijaté od vzniku požadavku na čtení souboru tohoto zařízení. Při zápisu CAN zpráv do souboru zařízení, jsou tyto data odeslána na sběrnici.

Pro zachování konzistence čtených a zapisovaných dat, bude nutné implementovat některý z principu synchronizace souborových operací.

Kapitola 8

Implementace řadiče sběrnice CAN

Programovou dokumentaci, vytvořenou pomocí nástroje Doxygen, lze nalézt na přiloženém CD. V této kapitole jsou shrnuty a stručně popsány hlavní funkční částí implementovaného řadiče.

8.1 Modul řadiče

Při vložení řadiče do jádra se zavolá inicializační funkce, která provede konfiguraci veškerého použitého hardware ve vestavěném systému, vytvoří soubory zařízení v adresáři `/dev`, zaregistruje jejich souborové operace a nastaví obsluhu přerušení od CAN sběrnice.

8.1.1 Inicializace hardware

PIO piny, které jsou využity ve vestavěném systému, jsou zapnuty a nastaveny podle potřeby řadiče.

SPI rozhraní AVR32 procesoru je nastaveno do režimu master a SPI rozhraní CAN sběrnice a RTC hodin jsou nastaveny do režimu slave. Konfigurace pro CAN a RTC se liší použitým způsobem časování, přesněji fázi a polaritou hodinového signálu. Při přepínání mezi jednotlivými zařízeními, pomocí signálu CS (chip select), se musí vždy nastavit zařízení do správného módu časování. Komunikace mezi připojenými zařízeními po SPI sběrnici probíhá pomocí základních metod pro čtení a zápis jednoho bytu.

Řadič CAN sběrnice, obvod MCP2515, se uvede do konfiguračního režimu. Poté se nastaví rychlost sběrnice implicitně na 125 kbps, na výběr jsou ještě volitelné rychlosti 78 kbps a 500 kbps, které lze nastavit pomocí parametru `canspeed` při vložení modulu do jádra. Dále jsou nastaveny filtry a masky, které jsou pro potřeby vestavěného systému zcela vypnuty, neboť řadič bude veškerou datovou komunikaci po CAN pouze směrovat do souboru zařízení umístěných. Poté je kontroler MCP2515 přepnut do zvoleného režimu, který je standardně nastaven na normální mód, ale je podporován i loopback mód, který lze zapnout pomocí parametru modulu `loopback`.

Modul RTC hodin je pouze okrajovou částí systému s CAN sběrnicí. V řadiči je proto jeho inicializace prováděna v obslužné funkci pro případ otevření souboru zařízení. Ovšem inicializace RTC je volána také případě, že při vložení modulu byl zadán parametr `datetime`, který očekává řetězec ve formátu `YY-MM-DDThh:mm:ss`.

8.1.2 Soubory zařízení

Při vložení modulu do jádra jsou v adresáři `/dev` vytvořeny soubory zařízení, pro CAN sběrnici zařízení `candev` a pro RTC modul zařízení `rtcdev`. Tato zařízení je možné číst, zapisovat do nich a samozřejmě otevřít či uzavřít. Tyto soubory jsou prostředky, které může používat jeden uživatelský proces a nebo mohou být sdílené mezi více procesy.

Data uchovávaná a šířená řadičem musí být konzistentní, proto jsou soubory zařízení chráněny semaforem, který vylučuje vícenásobné použití a v našem případě otevření souboru zařízení.

Při otevření určitého zařízení je proveden výběr odpovídajícího SPI rozhraní a otevřen semafor, který zabrání, aby aktuálně používané zařízení používal jiný proces. Po otevření je možné ze zařízení číst data a nebo je do něj zapisovat.

Při čtení souboru `rtcdev` je vrácenou hodnotou řetězec s aktuálním datem a časem. Při zápisu časových údajů ve formátu `YY-MM-DDThh:mm:ss`, je RTC modul nastaven na zadaný časový údaj. Pokud dojde k pokusu o zápis neplatných dat, je vrácena chyba.

Při čtení zařízení `candev` je přečtenou hodnotou aktuálně přijatá CAN zpráva. Při zápisu do zařízení, je očekávanou vstupní hodnotou CAN zpráva ukončená znakem `#`. Princip zpracování CAN zpráv je popsán podrobněji níže.

8.1.3 Zpracování CAN zpráv

V případě, že po CAN sběrnici přijde zpráva, vyvolá kontroler MCP2515 přerušení a v obslužné rutině tohoto přerušení dojde ke zpracování přijaté zprávy. Zpracování přijaté CAN zprávy je podmíněno nastavením příznaku čtení zařízení `candev`.

Pokud je tento příznak nastaven, přijatá zpráva je uložena do bufferu a jeho obsah je vrácen v průběhu čtení zařízení. V případě, že příznak nastaven není, jsou přijaté CAN zprávy zahozeny a při čtení zařízení je vrácen prázdný buffer.

V případě zápisu dat do zařízení `candev` je na vstupu očekávána CAN zpráva ukončená znakem `#`. Záměrem nutnosti ukončovat CAN zprávy speciálním znakem je, že možnost jejich hromadného zápisu a odeslání. Po zapsání dat do zařízení, je každá zpráva odeslána po CAN sběrnici.

8.2 Použití řadiče

Implementovaný programový řadič bude použit jako komunikační rozhraní mezi hardwarem vestavěného systému a uživatelských procesů, běžících ve spuštěném operačním systému.

8.2.1 Testovací software

Pro testování a demonstraci programového ovladače byly vyvinuty tyto aplikace (viz. příloha A):

canwtest

Tento testovací program čte soubor zařízení CAN v neblokujícím režimu a vypisuje přijaté zprávy na standardní výstup.

Hlavní funkcí tohoto programu je ale identifikace programu pomocí identifikační služby definované standardem CANAerospace (viz. 2.6). Ve firmě UNIS, a.s byl vyvinut software Monitor, který dokáže spravovat CAN zařízení, dokáže nastavovat a číst jejich registry

a sledovat přenos dat po CAN sběrnici. Monitor je přitom spuštěn na pc, připojeném na CAN sběrnici pomocí CAN/RS232 převodníku. Program canwtest dokáže tomuto programu o sobě odeslat informace, pokud o to Monitor požádal.

loopbacktest

Tento program slouží pro test funkčnosti řadiče v loopback režimu. Program po spuštění čeká na zadání zprávy ve formátu CANAerospace (viz. 2.6), kterou by mohl odeslat, poté v neblokujícím režimu čte zařízení CAN a vypisuje přijaté zprávy na standardní výstup.

Kapitola 9

Závěr

Tato bakalářská práce popisuje koncepci implementace fyzického řadiče CAN sběrnice a programových ovladačů pro jádro operačního systému Linux. CAN sběrnice je součástí vestavěného systému, postaveného na vývojové desce Atmel ATNGW100, ke které je připojena externí deska implementující CAN sběrnici a RTC modulem hodin. Přenos dat mezi zařízeními probíhá přes jejich SPI rozhraní. Tato komunikace je řízena programovým řadičem, jehož modul je vložen do jádra operačního systému. Tento řadič zprostředkovává komunikaci mezi hardwarem, jeho základním firmare a aplikačním prostorem. Operační systém je upraven pro specifické požadavky systému a zkompilován pro použitou architekturu AVR32 procesoru.

Při vložení modulu do jádra se vytvoří soubory zařízení, které poskytují uživatelským procesům možnost číst a zapisovat časové údaje do modulu reálných hodin a možnost číst nebo posílat CAN zprávy po sběrnici.

Vývoj tohoto vestavěného systému je zaštítěn brněnskou společností UNIS, a.s a jejím oddělením mechatronických systému. Tento systém zde bude použit jako prostředek pro monitorování vyvíjených automatizovaných zařízení a kritických řídicích aplikací. V operačním systému bude běžet monitorovací software, který bude pomocí protokolu CAN a implementovaného řadiče zpracovávat a sledovat data přicházející. po CAN sběrnici z monitorovaného systému.

Implementovaný programový ovladač řadiče sběrnice CAN je již plně použitelný, ale aby byla zaručena jeho stabilita, funkčnost a bezchybnost návrhu, musí být otestován na praktickém případu a tím podroben zátěžovému testu. Jeho vývoj bude dále pokračovat, v mnoha věcech se může funkčnost řadiče rozšířit a vylepšit. Potencionálně rozšířen může být rozsah podporovaných souborových operací se zařízeními. Dále může být zlepšena synchronizace při čtení a přijímaní zpráv po CAN sběrnici nebo synchronizace přístupu při provádění souborových operací se zařízením.

Literatura

[1] Deset let činnosti CiA – deset let rozvoje sběrnice CAN. *Automa*, ročník 8, č. 7, 2002.

[2] NGW100 Hardware reference. 2007, [online].
URL http://www.avrfreaks.net/wiki/index.php/Documentation:NGW/NGW100_Hardware_reference

[3] NRZ Encoding Definiton. 2008, [online].
URL http://www.interfacebus.com/NRZ_Definition.html

[4] ATNGW100 Network Gateway Kit. 2009, [online].
URL http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4102

[5] Atmel Corporation: *AVR32 32-bit Microcontroller AT32AP7000*. 2007.

[6] Atmel Corporation: *AVR32 Architecture Document*. 2007.

[7] Bosch, R.: *CAN specification*. Gmb H, druhé vydání, [cit. 2009-05-09], [online].
URL http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[8] Brake, C.: Flash Filesystems for Embedded Linux Systems. červenec 2001, [online].
URL http://www.linuxjournal.com/article/4678

[9] Catsoulis, J.: *Designing embedded hardware*. Sebastopol: O'Reilly, druhé vydání, 2005, ISBN 0-596-00755-8.

[10] Jelínek, L.: *Jádro systému Linux - Kompletní průvodce programátora*. Brno: Computer Press, a.s, první vydání, 2008, ISBN 978-80-251-2084-2.

[11] Maxim/Dallas Semiconductor: *DS1305 - Serial Alarm Real-Time Clock*. 2005, datasheet.

[12] Microchip Technology Inc.: *Stand-Alone CAN Controller With SPI Interface*. 2007, datasheet.

[13] Olaf Pfeiffer, C. K., Andrew Ayre: *Embedded Networking with CAN and CANopen*. Annabooks, Leden 2004, ISBN 0929392787.

[14] Stock, M.: *CAN Aerospace - Interface specification for airborne CAN applications*. Stock Flight Systems, revize 1.7 vydání, [cit. 2009-05-09], [online].
URL www.a2tech.eu/PDF/canas_17.pdf

[15] Texas Instruments: *3.3 V Can Transceivers*. 2009, datasheet.

[16] Yaghmour, K.: *Building embedded Linux systems*. Boston: O'Reilly, první vydání, 2003, ISBN 0-596-00222-X.

Dodatek A

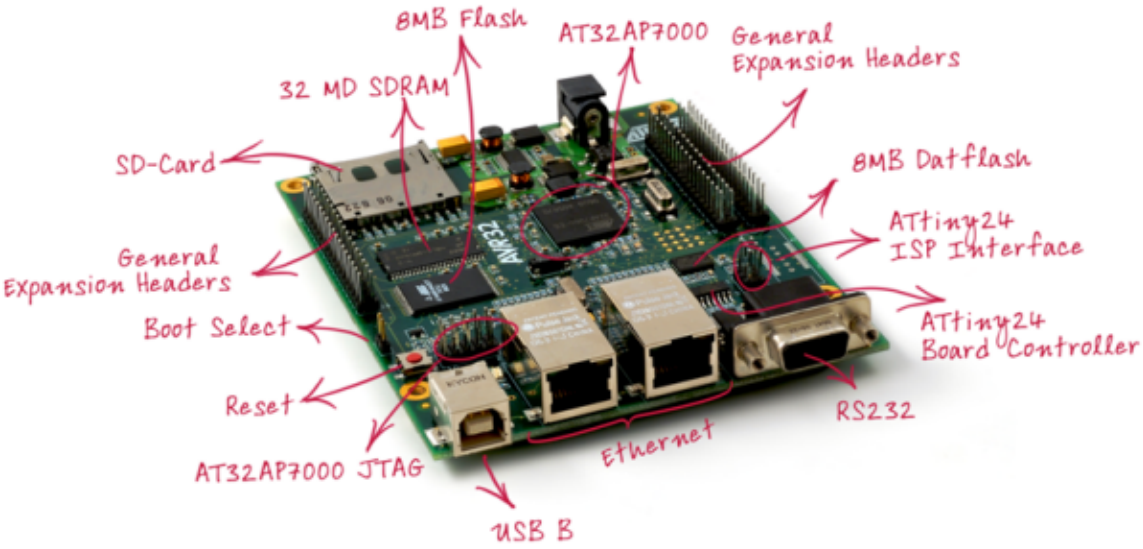
Obsah přiloženého CD

Přiložené CD obsahuje následující adresářovou struktůru:

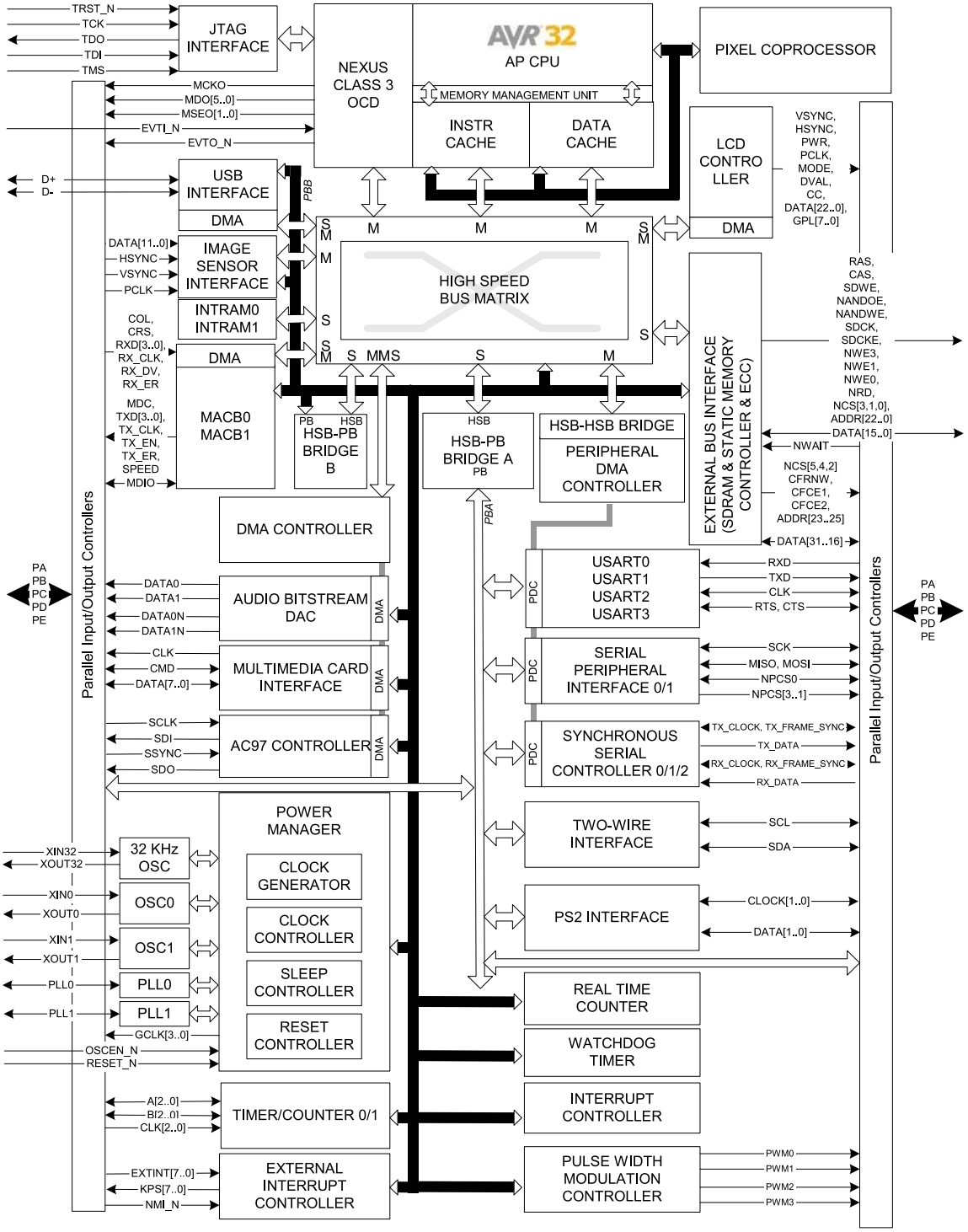
`/`	
`/datasheet/`	Datasheety použitých součástek a hardwárového vybavení.
`/manual/`	Manual pro zprovoznění vestavěného systému
`/readme.txt`	Obsah CD
`/schemes/`	Schémata zapojení externí desky
`/sources/`	
`/canw_driver/`	Zdrojové kódy řadiče CAN sběrnice
`/test/`	Zdrojové kódy testovacích aplikací
`/systems/`	
`/buildroot/`	Skripty potřebné pro vytvoření souborového systému
`/kernel/`	Upravené jádro operačního systému Linux
`/text/`	Text bakalářské práce ve formátu pdf a zdrojového kódu v LaTeXu

Dodatek B

Schémata



Obrázek B.1: Vývojový kit Atmel ATNGW100 [2]



Obrázek B.2: Blokové schéma AVR32 procesoru AP7000 [5]