České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídící techniky

DIPLOMOVÁ PRÁCE

Stanislav Marek Praha 2006

Abstrakt

Diplomová práce se zabývá návrhem malého zapouzdřeného zařízení se 16-ti bitovým procesorem vybaveného komunikací CANopen, popisem sběrnice CAN, popisem standardu CANopen a portací části programového vybavení pro práci s mikroprocesorem.

Za základ komunikační vrstvy CANopen a software komunikačního uzlu byl vybrán projekt CanFestival, který byl portován a přizpůsoben cílové platformě. Pro tento projekt byla napsána knihovna pro práci s rozhraním CAN na cílové platformě.

Abstract

This diploma thesis is focused on the development of embedded device providing CANopen communication protocol support for the 16 bit microcontroller. The diploma thesis describes CANopen communication protocol, CAN communication layer and port of the software equipment used by microcontroller.

The CanFestival project has been chosen as base of CANopen communication stack and slave node implementation. The project has been ported to the target CPU and integrated with its runtime support environment and required target CAN controller routines has been implemented.

Prohlášení

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

V Praze dne …… …….

podpis

Poděkování

Na tomto místě bych rád poděkoval zejména rodičům a i všem ostatním, kteří udržovali mé studijní odhodlání, měli se mnou trpělivost během těžkých dnů ve zkouškových obdobích a sdíleli se mnou radost z dosažených výsledků.

Velmi rád bych vyjádřil svou vděčnost vedoucímu diplomové práce ing. Pavlu Píšovi, který mi udělil mnoho cenných rad a věnoval svůj čas. Jeho připomínky přispěly nejen k této diplomové práci, ale i k nabytí zkušeností, které bych jinak musel dlouho získávat.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Cílem práce je návrh zařízení podporujícího protokol CANopen, protože v dnešní době stále více výrobců používá tento komunikační protokol pro svá zařízení a tudíž se uplatňuje v mnoha oblastech. Použití toho protokolu je standardizováno pro mnoho zařízení z oblastí automatizace, řízení, robotiky a elektronické výbavy automobilů. Předkládaná práce se zaměřuje na postup návrhu takovéhoto zařízení z hlediska výběru hardware a programovacích nástrojů. Dále práce zahrnuje realizaci CANopen slave zařízení a na závěr ukazuje možnost budoucího rozšíření práce a využití předkládaných poznatků.

Dnes velké množství projektů nabývá na složitosti zejména z několika následujících hledisek: bezpečnosti, uživatelské přístupnosti, použitelnosti a možnosti navázat na předchozí myšlenky. Při práci na projektu, který má mít delšího trvání než je doba psaní diplomové práce, je nutné zohlednit dosavadní výsledky, neopakovat předchozí chyby a dát ostatním možnost využít vlastních poznatků.

Práce je tématicky rozdělena na popis hardware, použitých komunikačních standardů, vývojových nástrojů, implementaci CAN knihovny a popis portovaného CANopen.

Důvody pro výběr mikroprocesoru společně s přehledem funkcí periférií a popisem vývojové desky jsou v následující kapitole. Další dvě kapitoly popisují standardy CAN sběrnice a CANopen protokolu.

Čtvrtá a pátá kapitola se zabývá použitými vývojovými nástroji a vysvětluje způsob kompilace projektu, konfiguraci HW a způsob přenosu aplikace do mikroprocesoru, včetně významu nejdůležitějších souborů zajišťujících běh aplikací.

CAN knihovna, která je jedním z cílů práce, je uvedena v šesté kapitole, kde je popsán způsob implementace a seznam příkazů, kterými lze ladit komunikaci po sběrnici CAN. Je zde také uveden seznam vytvořených příkazů a popis driveru pro CANopen. Způsob portace CANopen aplikace, která byla hlavním cílem předkládané práce je uveden v další kapitole, kde je popis jednotlivých souborů aplikace, způsob implementace časovačů a základní princip práce s komunikačními objekty.

2. Popis hardware

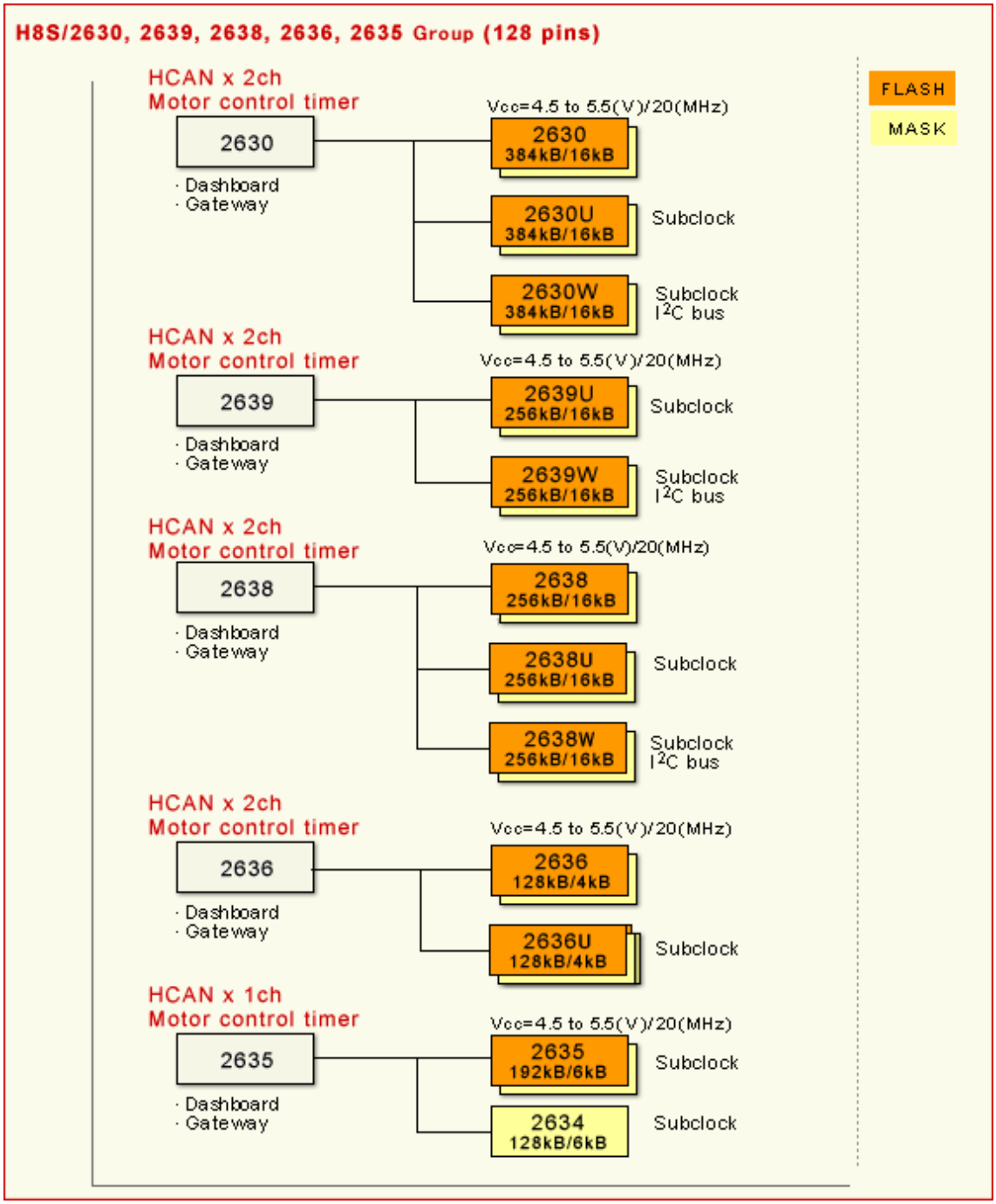
V této kapitole je popsán použitý hardware, vybraný mikroprocesor a použité periferie. Výběr mikroprocesoru byl proveden z hlediska ceny, z hlediska podpory tj. dostupnosti a využitelnosti programového prostředí a využití ve více projektech.

2.1 Mikroprocesor H8S/2638

Z hlediska využití ve více aplikačních oblastech byl vybrán mikroprocesor H8S/2638 od firmy Renesas. Pro komunikaci s okolními zařízeními je mikroprocesor vybaven rozhraními (CAN, SCI, I2C), dále nabízí periferie pro sběr dat a vstupně výstupní operace (A/D převodník a dig. vstupy a výstupy). Při mnoha aplikacích lze využít jednotku TPU obsahující časovače a čítače. Pro použití při řízení je k dispozici PWM kontrolér pro motory. Pro časově kritické přenosy dat může být výhodné využít integrovanou jednotku DTC. Mikroprocesor je zaměřen na jednodušší aplikace nepotřebující velký výpočetní výkon, ale díky množství periférií umožňuje velmi rozmanité použití.

Dalším hlediskem výběru mikroprocesoru bylo využití starších projektů s procesorem typu H8S/2633 a nástrojů pro kompilaci zdrojových kódů. Takto bylo využito kompilačního software na platformě Linux s licencí GNU a již připravených vývojových nástrojů.

2.1.1 Popis rodiny H8S



Obrázek 2.1: Vývojové větve rodiny H8S/2600.

Mikroprocesor je z rodiny řady H8S/2600, která je vyšší produktovou řadou. Celá řada těchto procesorů nabízí různé kombinace periferií a výkonné 16-ti bitové procesorové jádro.

2.1.2 Popis mikroprocesoru H8S/2638

Mikrokontrolér je zaměřen na použití v aplikacích vyžadujících širší výbavu periferiemi. Integruje 16-ti bitový časovač/generátor pulzů (TPU), programovatelný pulsní generátor (PPG), watchdog časovač (WDT), sériové komunikační rozhraní (SCI), A/D převodník, časovač pro řízení motorů (PWM), řadič pro hardwarové ladící breakpointy, vstupní a výstupní porty. Dále je vybaven vnitřním řadičem pro nezávislý přenos dat (DTC), který umožňuje rychlý přenos dat vyvolaný například přerušením, aniž by bylo potřeba účasti CPU. Na čipu se nachází také paměť typu FLASH a RAM.

Hlavní výbava:

- Operační frekvence /Podporované napájecí napětí
- 20 MHz/4,5V – 5,5V
- Velikost paměti 256kB FLASH a 16kB RAM
- Zabudované násobení 16-ti bitový x 16-ti bitový registr
- 2 x řadič sběrnice CAN
- 6-ti kanálový, 16-ti bitový časovač
- Řízení motorů, 16-ti kanálový PWM
- 3x sériový kanál
- 10-ti bitový A/D převodník s 12-ti kanály

Zabudované funkce:

- H8S/2600 základní jednotka je postavena na 16-ti bitové architektuře
- Zpětně kompatibilní s H8/300 a H8/300H CPU na objektové úrovni
- Šestnáct 16-ti bitových registrů
- 69 základních instrukcí
- Adresní prostor o velikosti 16MB
- Přerušovací systém
- 55 interních zdrojů přerušení, 7 externích
- Řadič sběrnice
- Šířka externí datové sběrnice 8/16 bitů
- Periferní zařízení
  - Řadič PC breakpointů, 2 kanály
  - Řadič přenosu dat (DTC), 85 kanálů
  - 16-ti bitový časovač/generátor pulzů, 6 kanálů
  - Programovatelný generátor pulzů (PPG), 8 kanálů
  - Watchdog časovač (WDT), 2 kanály

    - Řízení motoru pulsně šířkovou modulací (PWM), 16 kanálů
    - Sériové komunikační rozhraní (SCI), 3 kanály
    - Řadič komunikace CAN, 2 kanály
    - 10-ti bitový A/D převodník, 12 vstupů
    - D/A převodník 8b, 2 výstupy
    - Generátor hodinových impulzů
    - I^2C komunikační rozhraní, 2 kanály
- Hlavní vstupní a výstupní porty
    - 72 I/O pinů
    - 12 pinů jen jako vstupní

2.1.3 Popis řadiče CAN komunikace

Procesor je vybaven dvěma nezávislými řadiči CAN komunikace, každý řadič obsahuje svoji sadu konfiguračních registrů a 16 schránek pro příjem či odesílání zprávy doplněných o registry pro nastavení parametrů zprávy. Oba kanály jsou shodné a liší se jen polohou registrů v adresové paměti a výstupními piny. Pro šetření energií je možné zapínat každý kanál zvlášť a využívat módů pro snížení spotřeby.

Přehled registrů HCAN řadiče

- **Master control register** (MCR) – 8b registr, zajišťuje základní nastavení CAN
- **General Status Register** (GSR) – 8b registr, indikuje stav CAN řadiče
- **Bit Configuration Register** (BCR) – 16b registr, nastavení časování řadiče
- **Mailbox Configuration Register** (MBCR) – 16b registr, nastavení message objektů (schránek) na vysílání nebo příjem
- **Transmit Wait Register** (TXPR) – 16b registr, požadavek na vyslání zprávy a indikace stavu zprávy vysílání/nečinost
- **Tramsmit Wait Cancel Register** (TXCR) - 16b registr, požadavek na zrušení zprávy čekající na vyslání
- **Transmit Acknowledge Register** (TXACK) - 16b registr, indikuje správně odeslanou zprávu
- **Abort Acknowledge Register** (ABACK) - 16b registr, indikuje úspěšné zrušení odesílané zprávy
- **Receive Complete Register** (RXPR) - 16b registr, indikuje správné přijetí zprávy (datového rámce i žádosti o data)
- **Remote Request Register** (RFPR) - 16b registr, indikuje správné přijetí zprávy žádost o data

- **Interrupt Register** (IRR) - 16b registr, indikuje volání jednotlivých zdrojů přerušení
- **Mailbox Interrupt Mask Register** (MBIMR) - 16b registr, zakazuje/povoluje přerušení od jednotlivých schránek
- **Interrup Mask Register** (IMR) - 16b registr, zakazuje/povoluje přerušení od různých zdrojů přerušení příslušných CAN řadiči
- **Receive Error Counter** (REC) - 16b registr, počet chyb při příjmu zprávy
- **Transmit Error Counter** (TEC) - 16b registr, počet chyb při odesílání zprávy
- **Unread Message Status** Register (UMSR) - 16b registr, indikuje přepsání zprávy ve schránce novou zprávou aniž byla předchozí přečtena
- **Local Acceptance Filter Mask** (LAFML, LAFMH) – dva 16b registry, nastavují filtr příchozích zpráv pro schránku 0
- **Message Control** (MC0 – MC15) – osm osmi bitových registrů pro každou schránku pro nastavení parametrů zprávy
- **Message Data** (MD0 – MC15) - osm osmi bitových registrů pro každou schránku sloužící k uložení odesílaných nebo čtení přijatých dat zprávy
- **Module Stop Control Register** (MSTPCRC) – 8b registr, ovládající možnost vypnutí a zapnutí příslušného modulu, v našem případě se jedná o řadič CANu HCAN0 nebo HCAN1

Popis registrů Message Control

V registru MC se nastavují parametry zprávy tj. identifikační číslo pro standardní nebo rozšířený formát, počet datových bajtů a jedná-li se o normální data nebo o žádost o data.

Každý z registrů MC0 až MC15 se skládá z osmi osmi bitových registrů, které jsou umístěny v paměti RAM a nejsou hardwarem inicializovány.

MCx[1] obsahuje pouze údaj o délce zprávy viz Obrázek 2.2 a Tabulka 2.1.

bit:	7	6	5	4	3	2	1	0
					DLC3	DLC2	DLC1	DLC0
počáteční hod.:	*	*	*	*	*	*	*	*
R/W:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Obrázek 2.2: Registr MCx[1].

Bit 3: DLC3	Bit 2: DLC2	Bit 1: DLC1	Bit 0: DLC0	počet datových bajtů ve zprávě
0	0	0	0	0
			1	1
		1	0	2
			1	3
	1	0	0	4
			1	5
		1	0	6
			1	7
1	x	x	x	8

Tabulka 2.1: Počet bajtů zprávy.

MCx[2], MCx[3], MCx[4] jsou nevyužité.

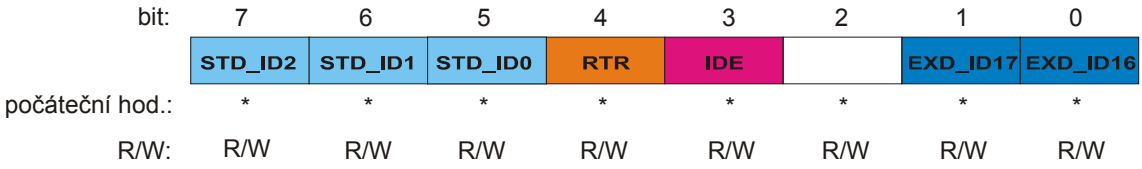
MCx[5], MCx[6], MCx[7], MCx[8] jsou registry obsahující informace o identifikátoru zprávy, rozlišení rozšířeného identifikátoru a rozlišení žádosti o data.

IDE – bit značící použití rozšířeného formátu identifikátoru

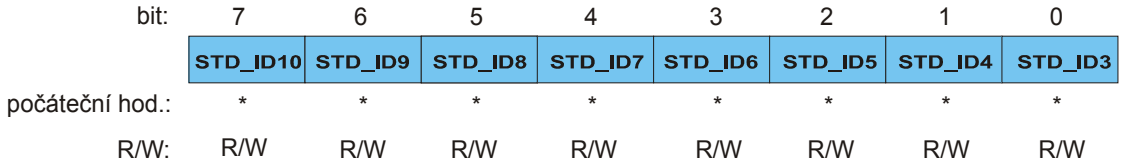
RTR – bit značící žádost o data

STD_ID0 – STD_ID10 – standardní identifikátor zprávy

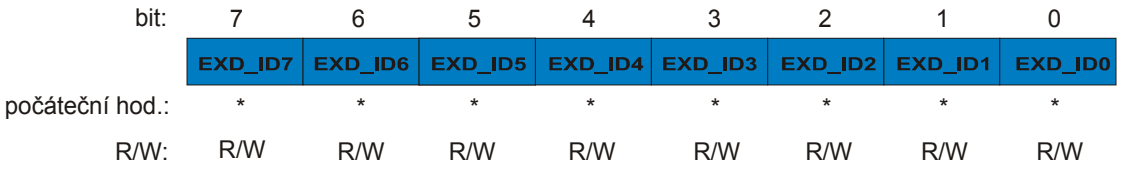
EXD_ID0 – EXD_ID17 – rozšířený identifikátor zprávy



Obrázek 2.3: Registr MCx[5].



Obrázek 2.4: Registr MCx[6].

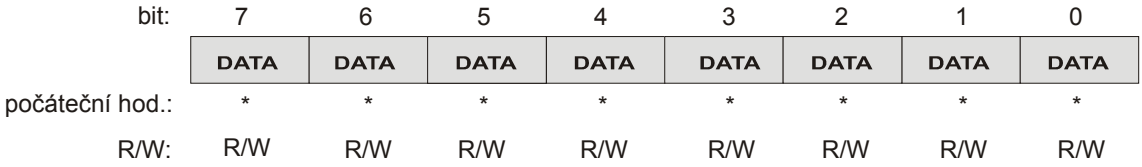


Obrázek 2.5: Registr MCx[7].

Obrázek 2.6: Registr MCx[8].

Popis registrů Message Data

V registrech MD se ukládají resp. čtou odesílaná resp. přijímaná data. Registry jsou uspořádány shodně jako registry MC. Přístup je opět osmibitový.

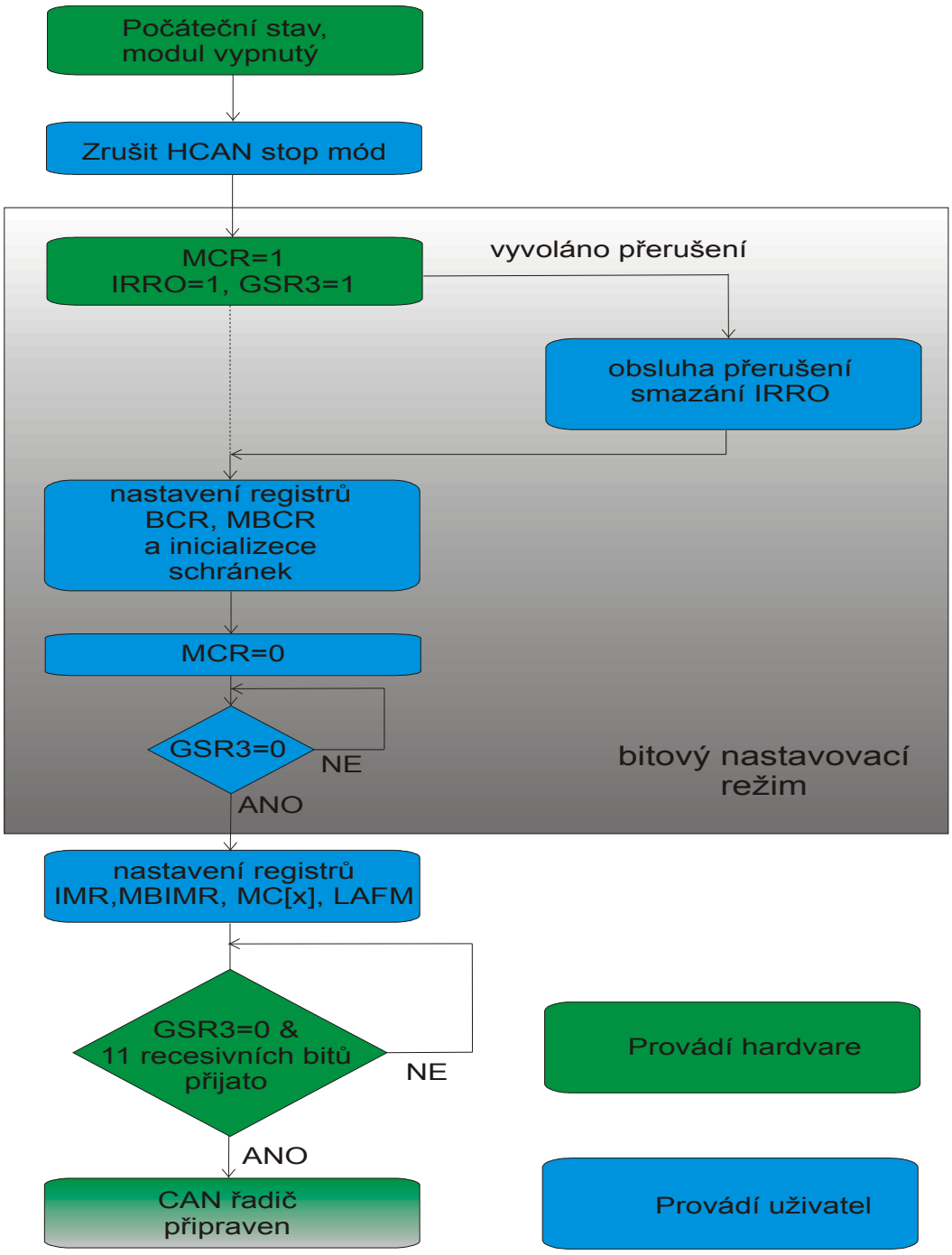


Obrázek 2.7: Registr MDx[x].

Inicializace CAN řadiče

Před použitím CAN řadiče je třeba provést jeho inicializaci. Při provedení HW resetu přejde řadič do módu module stop, což je stav kdy je řadiči vypnuto napájení a nejsou dostupné ani jeho registry.

Aby bylo možné zapnout napájení je nutné mít nakonfigurovánu obsluhu přerušení tj. napsanou obslužnou funkci, která je zaregistrována k příslušnému vektoru přerušení. Důvodem je skutečnost, že při zapnutí napájení HCAN řadiče je vždy vyvoláno přerušení. Postup jak nastavit registry viz Obrázek 2.8.



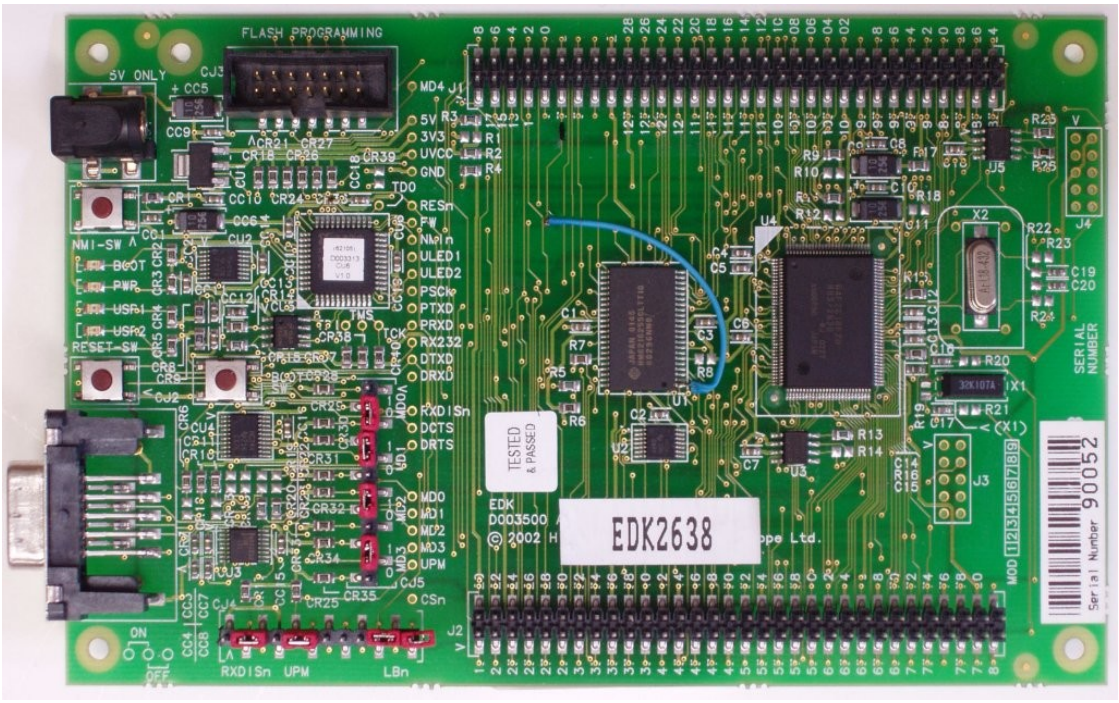
Obrázek 2.8: Postup nastavení řadiče CANu.

2.2 Vývojový kit od firmy Renesas

Jako vhodný HW byl vybrán vývojový kit od firmy Renesas s deskou EDK 2638 detaily v [5].

2.2.1 Výbava desky EDK 2638

- Mikroprocesor H8S/2638
- Paměť 256kB static RAM, úpravou adresování rozšířeno na 0,5MB
- Převodník na RS232
- Budiče sběrnice CAN
- 2 xLED pro uživatelské použití
- Přepínače pro konfiguraci procesoru a testovací výstupy



Obrázek 2.9: Vývojová deska EDK 2638.

2.2.2 Rozšíření paměti

Vývojová deska je z výroby vybavena pamětí HM62255HCLTT-10. Velikost paměti RAM uvedená v manuálu k desce je 256kB. Při testech paměti jsem zjistil, že data zapsaná v 8-mi bitovém režimu zápisu do paměti nelze přečíst v 16-ti bitovém režimu čtení z paměti. To vedlo k rozboru typu paměti a zjištění, že paměť je velikosti 512-kB (256kB x 16). Chybou v návrhu zapojení paměti dochází, při použití rozdílného způsobů zápisu a čtení, ke čtení jiných paměťových buněk, než do kterých bylo zapsáno.

Provedl jsem úpravu při níž se připojil k paměti další adresní bit A18 procesoru na místo stávající adresy A0 na paměti viz Tabulka 2.2 Nyní je možné provádět čtení a zápis v šestnácti bitovém i osmibitovém režimu.

využitelná velikost vnější RAM	µprocesor	paměť	režim přístupu	
256kB	pin č. 16	pin č. 1	osmi/šestnácti bitový	dříve
512kB	pin č. 7	pin č. 1	16-ti bitový	nyní

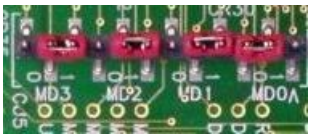
Tabulka 2.2: Zapojení a způsob přístupu do vnější paměti RAM.

Význam propojek na desce

Na konektoru CJ5 jsou vyvedeny piny procesoru nastavující operační mód. Jumpery pro bity MD0 až MD2 nastavují způsob přístupu do paměti viz [3]. Jumper MD3 není využit. Detail propojovacího pole Obrázek 2.11.

Na konektoru CJ4 se nastavuje vypnutí RS232 pomocí RXDISn a aktivace uživatelského programovacího režimu pomocí UPM. LBn odpojuje signál přenosu nižšího bajtu do paměti. Detail viz Obrázek 2.10.

Obrázek 2.10: Konektor CJ4



Obrázek 2.11: Konektor CJ5

3. Komunikace CANopen

CANopen je vyšší komunikační protokol vytvořený na základě sběrnice CAN (*Controler Area Network*). Jedná se o široce používaný protokol v mnoha odvětvích např. lékařských zařízeních, automobilním průmyslu, námořních systémech, veřejné dopravě, automatizaci atd.

Nejprve bude popsán protokol CAN tj. srovnání s ISO/OSI modelem, používaný hardware, jednotlivé vrstvy a datové rámce.

V další části bude uveden popis protokolu CANopen srovnání s ISO/OSI modelem, struktura aplikace, komunikační objekty a metody synchronizace a specifikace pro vytváření zařízení DS 301 a DSP 402.

3.1 Popis protokolu CAN

Controller Area Network (CAN) je sériový komunikační protokol detaly v [2], který byl původně vyvinut firmou Bosch pro nasazení v automobilech. Vzhledem k tomu, že přední výrobci integrovaných obvodů implementovali podporu protokolu CAN do svých produktů, dochází k stále častějšímu využívání tohoto protokolu i v různých průmyslových aplikacích. Důvodem je především nízká cena, snadné nasazení, spolehlivost, vysoká přenosová rychlost, snadná rozšiřitelnost a dostupnost potřebné součástkové základny.

V současné době má protokol CAN své pevné místo mezi ostatními procesními sběrnicemi a je definován normou ISO 11898, která popisuje fyzickou vrstvu protokolu a specifikaci CAN 2.0A. Varianta specifikace CAN 2.0B rozšiřuje specifikaci o rámce s rozšířeným identifikátorem a zavádí nové dva pojmy – standardní a rozšířený formát zprávy (lišící se v délce identifikátoru zprávy). Tyto dokumenty definují pouze fyzickou a linkovou vrstvu protokolu podle referenčního modelu ISO/OSI. Aplikační vrstva protokolu CAN je definována několika vzájemně nekompatibilními standardy (CAL/CANopen, DeviceNet, ...).

3.1.1 Vlastnosti protokolu CAN

CAN byl navržen tak, aby umožnil provádět distribuované řízení systémů v reálném čase s přenosovou rychlostí do 1Mbit/s a vysokým stupněm zabezpečení přenosu proti chybám. Jedná se o protokol typu *multi-master*, kde každý uzel sběrnice může být *master* a může iniciovat přenos zpráv a předávat data jiným uzlům. Pro řízení přístupu k médiu je použita metoda s náhodným přístupem a příposlechem komunikace, která řeší kolize na základě prioritního rozhodování. Po sběrnici probíhá komunikace mezi dvěma uzly pomocí zpráv (datová zpráva a žádost o data), a management sítě (signalizace chyb, pozastavení komunikace) je zajištěn pomocí dvou speciálních zpráv (chybové zprávy a zprávy o přetížení).

Zprávy vysílané po sběrnici protokolem CAN neobsahují žádnou informaci o cílovém uzlu, kterému jsou určeny, a jsou přijímány všemi ostatními uzly připojenými ke sběrnici. Každá zpráva je uvozena identifikátorem, který udává význam přenášené zprávy a její prioritu. Nejvyšší prioritu má zpráva s identifikátorem 0. Protokol CAN zajišťuje, aby zpráva s vyšší prioritou byla v případě kolize dvou zpráv doručena přednostně a dále je možné na základě identifikátoru zajistit, aby uzel přijímal pouze ty zprávy, které se ho týkají (*Acceptance Filtering*).

3.1.2 Fyzická vrstva

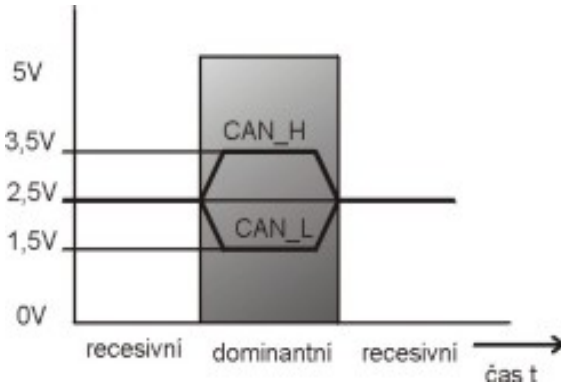
Fyzická vrstva definuje jak jsou po sběrnici přenášeny signály. Nedefinuje však přenosové médium a napěťové úrovně přenosu.

Základním požadavkem na fyzické přenosové médium protokolu CAN je, aby realizovalo funkci logického součinu. Standard protokolu CAN definuje dvě vzájemně komplementární hodnoty bitů na sběrnici *dominant a recessive*. Jedná se v podstatě o jakýsi zobecněný ekvivalent logických úrovní, jejichž hodnoty nejsou určeny a skutečná reprezentace záleží na konkrétní realizaci fyzické vrstvy. Pravidla pro stav na sběrnici jsou jednoduchá a jednoznačná. Vysílají-li všechny uzly sběrnice *recessive* bit, pak na sběrnici je úroveň *recessive*. Vysílá-li alespoň jeden uzel *dominant* bit, je na sběrnici úroveň *dominant*.

3.1.3 Použitá fyzická vrstva HI-SPEED CAN

Napěťové úrovně sběrnice CAN

Sběrnice CAN používá pro odlišení logických úrovní bitů dvou napěťových úrovní nazvaných dominantní a recesivní, viz Obrázek 3.1



Obrázek 3.1: Napěťové úrovně sběrnice CAN.

Recesivní úroveň je definována nulovým rozdílem potenciálů mezi oběma vodiči CAN sběrnice, dominantní úroveň je definována napětím vodiče CAN_H 3,5 V proti zemnímu vodiči a napětím vodiče CAN_L 1,5 V proti zemnímu vodiči. Pro eliminaci odrazů na vedení je sběrnice na obou koncích přizpůsobena zakončovacími odpory o velikosti 120 Ω.

Přesné elektrické charakteristiky sběrnice jsou detailně popsány v normě ISO 11898, zde se omezím na pouhé shrnutí několika nejdůležitějších a pro praktické využití užitečných faktů týkajících se fyzické vrstvy protokolu CAN. Přehled některých parametrů uvádí Tabulka 3.1 Na sběrnici může být teoreticky připojeno neomezené množství uzlů, avšak s ohledem na zatížení sběrnice a zajištění správných statických i dynamických parametrů sběrnice norma uvádí jako maximum 30 uzlů připojených na sběrnici. Maximální délka sběrnice je pro přenosovou rychlost 1Mbit/s udána normou 40m. Maximální délky sběrnice pro různé přenosové rychlosti uvádí Tabulka 3.1. Kromě délky pro rychlost 1 Mbit/s dané normou jsou ostatní délky pouze informativní a závisí na mnoha parametrech (např. typu použitého kabelu).

Přenosová rychlost	Maximální délka sběrnice
1 Mbit/s	40 m
500 kbit/s	112 m
300 kbit/s	200 m
100 kbit/s	640 m
50 kbit/s	1340 m
20 kbit/s	2600 m
10 kbit/s	5200 m

Tabulka 3.1: Závislost délky sběrnice na rychlosti.

3.1.4 Linková vrstva

Linková vrstva protokolu CAN je tvořena dvěma podvrstvami - MAC a LLC: První z nich má na starosti přístup k médiu MAC (*Medium Access Control*) a jejím úkolem je provádět kódování dat, vkládat doplňkové bity do komunikace (*Stuffing/Destuffing*), řídit přístup všech uzlů k médiu s rozlišením priorit zpráv, detekce chyb a jejich hlášení a potvrzování správně přijatých zpráv. Druhou podvrstvou linkové vrstvy je LLC (*Logical Link Contol*), která má za úkol provádět filtrování přijatých zpráv (*Acceptance Filtering*) a hlášení o přetížení (*Overload Notification*).

3.1.5 Zabezpečení přenášených dat a detekce chyb

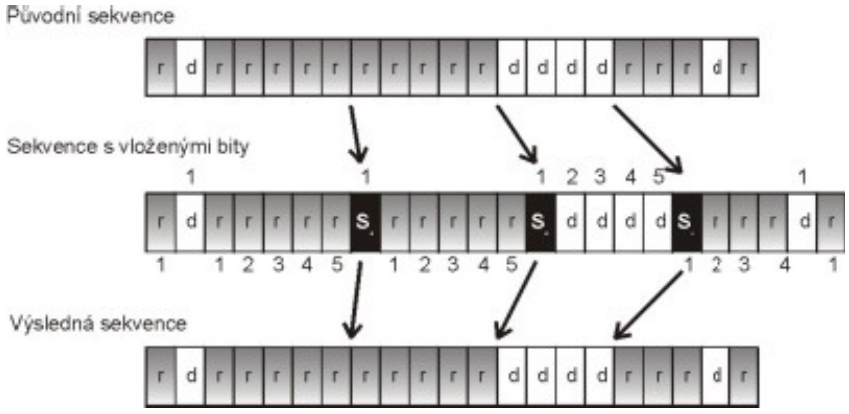
Zprávy přenášené pomocí protokolu CAN jsou zabezpečeny několika mechanizmy, které jsou v činnosti současně.

Monitoring

Již zmiňovaná metoda, kdy vysílač porovnává hodnotu právě vysílaného bitu s úrovní detekovanou na sběrnici. Jsou-li obě hodnoty stejné, vysílač pokračuje ve vysílání. Pokud je na sběrnici detekována jiná úroveň než odpovídá úrovni vysílaného bitu, a probíhá-li právě řízení přístupu na sběrnici (vysílá se Arbitration Field), přeruší se vysílání a přístup k médiu získá uzel vysílající zprávu s vyšší prioritou. Pokud je rozdílnost vysílané a detekované úrovně zjištěna jinde než v Arbitration Field a v potvrzení přijetí zprávy (ACK Slot), je vygenerována chyba bitu.

CRC kód (Cyclic Redundancy Check)

Na konci každé zprávy je uveden 15-ti bitový CRC kód, který je generován ze všech předcházejících bitů příslušné zprávy podle polynomu: $x_{15} + x_{14} + x_{10} + x_8 + x_7 + x_4 + x_3 + 1$. Je-li detekována chyba CRC libovolným uzlem na sběrnici, je vygenerována chyba CRC.



Obrázek 3.2: Vkládání bitů.

Vkládání bitů (Bit stuffing)

Vysílá-li se na sběrnici pět po sobě jdoucích bitů jedné úrovně, je do zprávy navíc vložen bit opačné úrovně viz Obrázek 3.2. Toto opatření slouží jednak k detekci chyb ale také ke správnému časovému sesynchronizování přijímačů jednotlivých uzlů. Je-li detekována chyba vkládání bitů, je vygenerována chyba vkládání bitů.

Kontrola zprávy (Message Frame Check)

Zpráva se kontroluje podle formátu udaného ve specifikaci a pokud je na nějaké pozici bitu zprávy detekována nepovolená hodnota, je vygenerována chyba rámce (formátu zprávy).

Potvrzení přijetí zprávy (Acknowlege)

Je-li zpráva v pořádku přijata libovolným uzlem, je toto potvrzeno změnou hodnoty jednoho bitu zprávy (ACK). Vysílač vždy na tomto bitu vysílá úroveň recessive a detekuje-li úroveň dominant, pak je přenos považován za úspěšný. Potvrzování přijetí zprávy je prováděno všemi uzly připojenými ke sběrnici bez ohledu na zapnuté či vypnuté filtrování zpráv (*Acceptance Filtering*).

3.1.6 Datové rámce

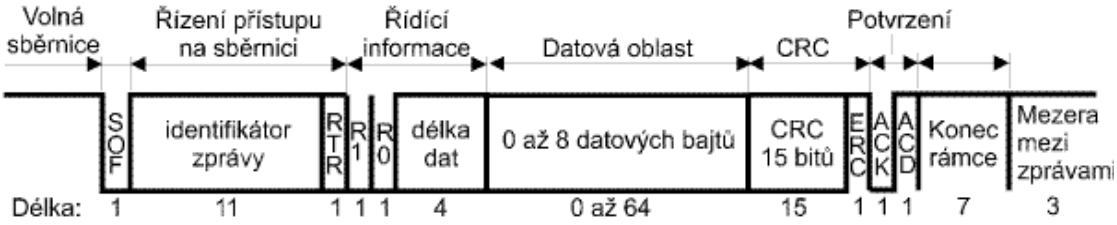
Specifikace protokolu CAN definuje čtyři typy zpráv.

První dva se týkají datové komunikace po sběrnici. Je to jednak datová zpráva, která představuje základní prvek komunikace uzlů po sběrnici, a dále pak zpráva na vyžádání dat, kdy uzel žádá ostatní účastníky na sběrnici o zaslání požadovaných dat. Datová zpráva umožňuje vyslat na sběrnici 0 až 8 datových bajtů. Pro jednoduché příkazy uzlům (např. příkazy typu vypni/zapni) není nutné přenášet žádné datové bajty (význam příkazu je dán identifikátorem zprávy), což zkracuje dobu potřebnou k přenosu zprávy a zároveň zvětšuje propustnost sběrnice, zvláště pak při silném zatížení. Zpráva na vyžádání dat je vyslána uzlem, který požaduje zaslání určitých dat. Odpovědí na tento požadavek je odeslání požadovaných dat uzlem, který má tato data k dispozici.

Poslední dvě zprávy (chybová zpráva a zpráva o přetížení) slouží k managementu komunikace po sběrnici, konkrétně k signalizaci detekovaných chyb, eliminaci chybných zpráv a vyžádání prodlevy v komunikaci.

Datová zpráva (Data Frame)

Protokol CAN používá dva typy datových zpráv. První typ je definován specifikací 2.0A a je v literatuře označován jako standardní formát zprávy (Standard Frame), zatímco specifikace 2.0B definuje navíc tzv. rozšířený formát zprávy (Extended Frame). Jediný podstatný rozdíl mezi oběma formáty je v délce identifikátoru zprávy,



Obrázek 3.3: Datový rámec CAN 2.0A.

která je 11 bitů pro standardní formát a 29 bitů pro rozšířený formát. Oba dva typy zpráv mohou být používány na jedné sběrnici, pokud je použitými řadiči podporován protokol 2.0B. Řadič standartu 2.0A nemůže rozšířené zprávy přijímat, ale neoznačuje je na rozdíl od starší implementace 1.0 za chybné.

Vyslání datové zprávy je možné pouze tehdy, je-li sběrnice volná (stav Bus Free). Jakmile uzel, který má připravenu zprávu k vyslání, detekuje volnou sběrnici, začíná vysílat. Zda získá přístup na sběrnici či nikoliv, záleží na již popsaném mechanizmu řízení přístupu k médiu. Strukturu datové zprávy podle specifikace 2.0A ilustruje Obrázek 3.3.

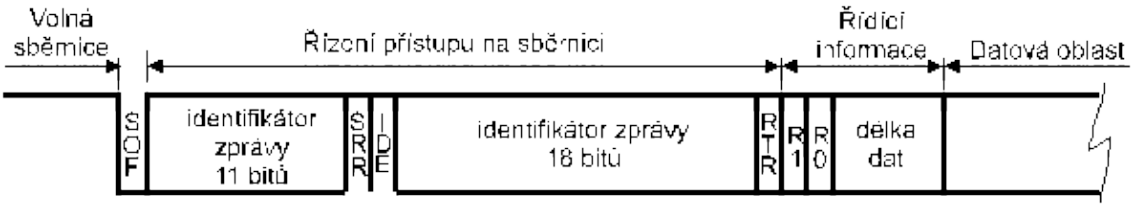
Význam jednotlivých částí datové zprávy podle specifikace 2.0A

- Začátek zprávy (SOF = Start Of Frame)
  - 1 bit dominant
- Řízení přístupu na sběrnici (Arbitration Field) - určení priority zprávy
  - Identifikátor zprávy - 11 bitů, udává význam přenášené zprávy
  - RTR bit (Remote Request) – 1 bit, příznak udává, zda se jedná o datovou zprávu nebo o žádost o vyslání dat. V datové zprávě musí být tento bit dominant, v žádosti o data recessive.
- Řídící informace (Control Field)
  - R0, R1 - rezervované bity
  - Délka dat – 4 bity, počet přenášených datových bajtů ve zprávě. Povolené hodnoty jsou 0 až 8.
- Datová oblast (Data Field) - datové bajty zprávy
  - Maximálně 8 bajtů je vysláno od MSB CRC (CRC Field) - 16 bitů, zabezpečovací CRC kód
- CRC kód – 15 bitů
- ERC (CRC oddělovač)
  - 1 bit recessive
- Potvrzení (ACK Field)
  - 2 bity
- ACK (bit potvrzení)
  - 1 bit
- ACD (oddělovač potvrzení)
  - 1 bit recessive
- Konec zprávy (End Of Frame)
  - 7 bitů recessive
- Mezera mezi zprávami (Interframe Space) - odděluje dvě zprávy
  - 3 bity recessive

Specifikace CAN 2.0B

Definuje dva formáty datové zprávy - standardní a rozšířený.

Standardní zpráva (Standard Frame) je převzata ze specifikace 2.0A, má délku identifikátoru zprávy 11 bitů. Jediným rozdílem je zde využití bitu R1 na indikaci, zda se jedná o rámec standardní nebo rozšířený. Zde se podle CAN 2.0B tento bit označuje IDE (Identifier Extended) a je dominant pro standardní formát a recessive pro rozšířený formát zprávy.



Obrázek 3.4: Rozšířený datový rámec CAN 2.0B.

Rozšířený rámec (Extended Frame) viz Obrázek 3.4 používá celkem 29 bitový identifikátor zprávy. Ten je rozdělen do dvou částí o délkách 11 (stejný identifikátor je použit ve standardním formátu) a 18 bitů. Bit RTR (Remote Request) je zde nahrazen bitem SRR (Substitute Remote Request), který má v rozšířeném formátu vždy hodnotu recessive. To zajišťuje, aby při vzájemné kolizi standardního a rozšířeného formátu zprávy na jedné sběrnici se stejným 11-ti bitovým identifikátorem, získal přednost standardní rámec. Bit IDE (Identifier Extended) má vždy recessive hodnotu. Bit (RTR) udávající, zda se jedná o datovou zprávu nebo žádost o data, je přesunut za konec druhé části identifikátoru. Pro řízení přístupu k médiu jsou použity ID (11 bit), SRR, IDE, ID (18 bit), RTR. V tomto pořadí je určena priorita datové zprávy.

Žádost o data (Remote Frame)

Formát žádosti o data je podobný jako formát datové zprávy. Pouze je zde RTR bit (pole řízení přístupu na sběrnici) nastaven do úrovně recessive a chybí datová oblast. Pokud nějaký uzel žádá o zaslání dat, nastaví takový identifikátor zprávy, jako má datová zpráva, jejíž zaslání požaduje Tím je zajištěno, že pokud ve stejném okamžiku jeden uzel žádá o zaslání dat a jiný data se stejným identifikátorem vysílá, přednost v přístupu na sběrnici získá uzel vysílající datovou zprávu, neboť úroveň RTR bitu datové zprávy je dominant a tudíž má tato zpráva vyšší prioritu.

Chybová zpráva (Error Frame)

Chybová zpráva slouží k signalizaci chyb na sběrnici CAN. Jakmile libovolný uzel na sběrnici detekuje v přenášené zprávě chybu (chyba bitu, chyba CRC, chyba vkládání bitů, chyba rámce), vygeneruje ihned na sběrnici chybový rámec. Podle toho, v jakém stavu pro hlášení chyb se uzel, který zjistil chybu, právě nachází, generuje na sběrnici buď aktivní (šest bitů dominant) nebo pasivní (šest bitů recessive) příznak chyby. Při generování aktivního příznaku chyby je přenášená zpráva poškozena (vzhledem k porušení pravidla na vkládání bitů), a tedy i ostatní uzly začnou vysílat chybové zprávy. Hlášení chyb je pak indikováno superpozicí všech chybových příznaků, které vysílají jednotlivé uzly. Délka tohoto úseku může být minimálně 6 a maximálně 12 bitů.

Po vyslání chybového příznaku vysílá každá stanice na sběrnici bity recessive. Zároveň detekuje stav sběrnice a jakmile najde první bit na sběrnici ve stavu recessive, vysílá se dalších sedm bitů recessive, které plní funkci oddělovače chyb (ukončení chybové zprávy).

Zpráva o přetížení (Overload Frame)

Zpráva o přetížení slouží k oddálení vyslání další datové zprávy nebo žádosti o data. Zpravidla tento způsob využívají zařízení, která nejsou schopna kvůli svému vytížení přijímat a zpracovávat další zprávy. Struktura zprávy je podobná zprávě o chybě, ale její vysílání může být zahájeno po konci zprávy (End of Frame), oddělovače chyb nebo předcházejícího oddělovače zpráv přetížení.

3.2 Popis CANopen

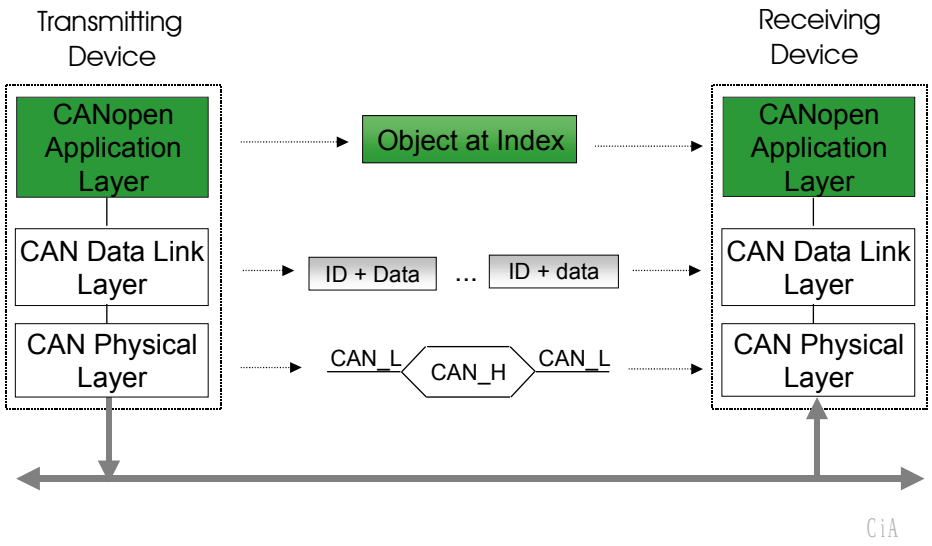
Síťový protokol CANopen, vycházející z komunikačního protokolu CAN a využívající hardware specifikovaný v ISO 11898, je specifikován organizací CiA (CAN in Automation).

Specifikace CANopen obsahuje aplikační vrstvu a komunikační profily (CiA DS-301) jako základní část pro programovatelná zařízení obsahující požadavky na kabely, konektory a jednotky soustavy SI. Standardizované profily zařízení jsou definovány v CiA DS-4XX.

Tento protokol byl původně navržen pro systémy v oblasti robotiky, ale díky možnostem specifikovat vlastní profily zařízení, se rychle rozšířil i do jiných oblastí.

3.2.1 Srovnání s ISO/OSI modelem

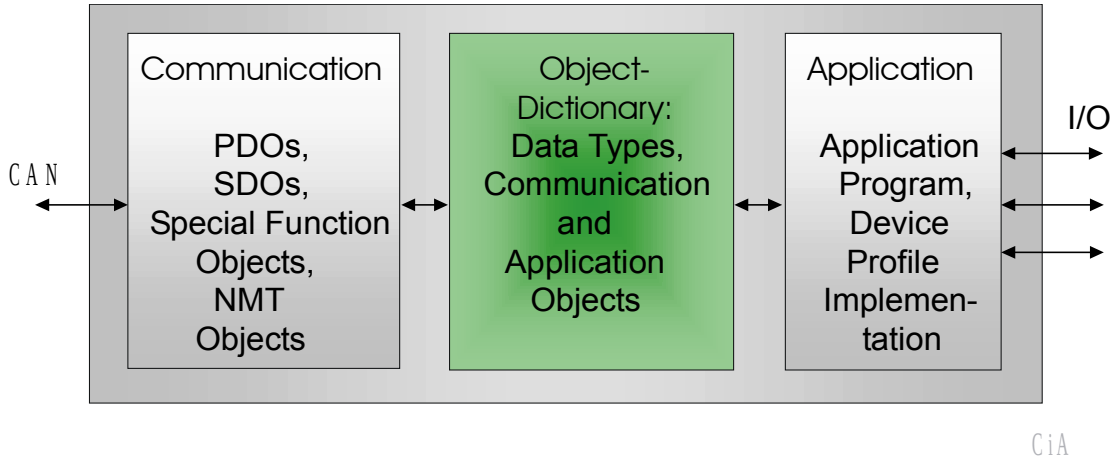
Vrstva CANopen odpovídá úrovni aplikační vrstvy ISO/OSI modelu viz Obrázek 3.5.



Obrázek 3.5: Vrstvy protokolu CANopen.

Linková a fyzická vrstva dle ISO/OSI modelu je popsána v části věnující se popisu sběrnice CAN.

3.2.2 Model CANopen zařízení



Obrázek 3.6: Model CANopen zařízení.

Zařízení je rozděleno na tři části:

- komunikační rozhraní a komunikační software zajišťují služby odesílání a přijímání komunikačních objektů na sběrnici
- slovník zařízení popisuje všechny datové typy, komunikační objekty a aplikační objekty používané tímto zařízením
- procesní rozhraní zajišťuje interní funkce a tvoří rozhraní pro uživatelskou aplikaci, která může pracovat s připojeným HW

3.2.3 Slovník zařízení OD

Slovník zařízení OD (Object Dictionary) sdružuje všechny objekty dostupné přes síť a definuje jejich počáteční nastavení. Každý objekt je adresován 16-ti bitovým indexem a osmi bitovým subindexem. Tvorba slovníku pro určitý typ zařízení je vázána standardy, které musí výrobce dodržovat. Rozložení informací na indexech definuje Tabulka 3.2.

Index	Objekt
0x0000	Rezervován
0x0001-0x0x001F	Statické Datové Typy
0x0020-0x0x003F	Komplexní Datové Typy
0x0040-0x0x005F	Výrobcem definované Datové Typy
0x0060-0x007F	Profilem zařízení definované Statické Typy
0x0080-0x009F	Profilem zařízení definované Komplexní Typy
0x00A0-0x0FFF	Rezervováno pro budoucí využití
0x1000-0x1FFF	Oblast Profilu Komunikace
0x2000-0x5FFF	Výrobcem využitá oblast
0x6000-0x9FFF	Profilem zařízení definovaná oblast
0x000-0xFFFF	Rezervované pro budoucí využití

Tabulka 3.2: Umístění objektů ve slovníku zařízení.

3.2.4 Komunikační objekty

CANopen protokol pro správu, zasílání dat a řízení komunikace po síti používá následující komunikační objekty.

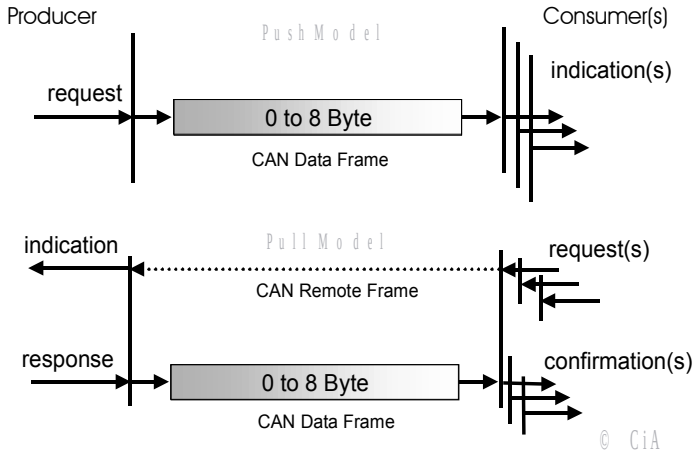
- Process Data Objects (PDO) – pro přenos důležitých real-time dat
- Service Data Objects (SDO) – pro zápis a čtení položek slovníku zařízení
- Speciální objekty
  - Synchronyzation (SYNC) – synchronizace zařízení
  - Time Stamp – zasílání časových značek
  - Emergency (EMCY) – nouzové zprávy
- Objekty pro správu sítě
  - NMT Message Object – konfigurace sítě
  - Boot-Up Object – zpráva o zapnutí zařízení
  - Error Control Object – posílání chybových zpráv

3.2.5 Metody komunikace

Metoda komunikace původce/spotřebitel

Využívá možnosti sběrnice CAN zaslat jedinou zprávu, kterou přijmou všechny uzly v síti viz Obrázek 3.7. Záleží na nastavení každého zařízení, zda-li zprávu přijme nebo odmítne. Tato schopnost uzlu přijmout či nepřijmout zprávu je Acceptance Filtering a lze jí realizovat nastavením řadiče CAN.

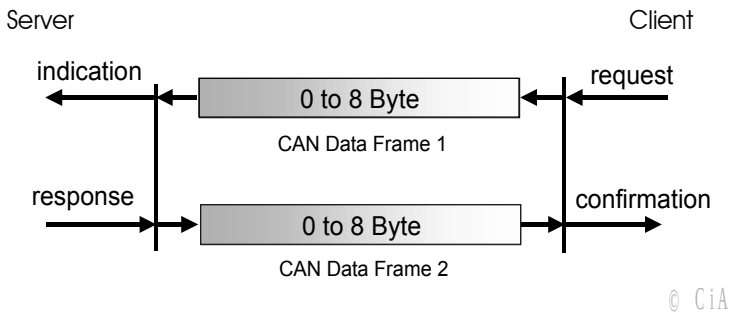
Model původce/spotřebitel dovoluje zařízením vysílat zprávy (push model) nebo požadovat zprávy (pull model).



Obrázek 3.7: Metoda komunikace původce/spotřebitel.

Metoda komunikace klient/server

Při klasické metodě komunikace typu klient/server vysílá klient požadavek, který serverem zodpovězen, čímž klient dostane potvrzení viz Obrázek 3.8.

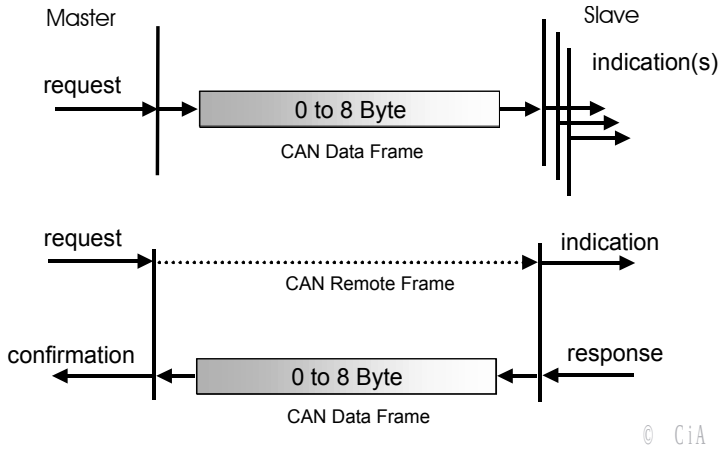


Obrázek 3.8: Metoda komunikace klient/server.

Metoda komunikace klient/server se používá pro přenos dat delších než 8 bajtů. Data jsou rozdělena do segmentů a po těchto segmentech přenášena s tím samým identifikátorem. Každá skupina segmentů z celkové zprávy je potvrzována příjemcem. Služby používající klient/server komunikaci umožňují čtení, zápis a zrušení přenosu.

Metoda komunikace master/slave

Může být iniciována pouze ze strany mastera, slave vždy čeká na požadavek vyslaný masterem.



Obrázek 3.9: Metoda komunikace master/slave.

Na sběrnici založené na CAN se master/slave komunikace realizuje pomocí volby příslušného identifikátoru. Nepotvrzovaná komunikace master/slave umožňuje také broadcasting.

3.2.6 Specifikace DS 301

Každá zařízení používající název CANopen by mělo být v souladu s touto normou, protože tato specifikace je základní pro každé zařízení pracující s CANopen a obsahuje definice všech potřebných součástí viz [7].

Definuje fyzickou a linkovou vrstvu CAN komunikace a komunikační rámce.

Jsou zde popsány základní a rozšířené datové typy včetně jejich kombinací.

Základem CANopen je použití standardních komunikačních objektů, které jsou definovány pro aplikační vrstvu. Norma určuje způsob přenosu dat, stanovení iden-

tifikátorů pro sběrnici CAN, kontrolu funkčnosti jednotlivých zařízení, zasílání nouzových objektů a jiné. Významným objektem je PDO přenášející časově kritická data.

3.2.7 Specifikace DSP 402

Pro variabilní použití CANopen standardu je potřeba vhodným způsobem rozšířit možnosti zařízení určených normou DS 301.

Účelem normy DSP 402 [8] je vytvořit nadstavbu pro zařízení z oblasti robotiky a pohonů. Norma definuje:

- údaje o zařízení, výrobce, verze, servisní prohlídky a další
- řídící proměnné
- převody mezi interními jednotkami a fyzikálními veličinami
- nastavení pro pozice zařízení a pohybu
- informace o základní poloze
- údaje pro určení pozice a pohybu
- parametry interpolace
- nastavení rychlostí
- nastavení momentů
- rychostní režim a jiné

4. Vývojové nástroje

Tato kapitola je věnována použití kompilačního systému a programového vybavení pro nahrávání programu do mikroprocesoru.

Pro kompilaci aplikace ze zdrojových kódů pro tento mikroprocesor lze použít více způsobu kompilace. Komerční prostředí dodávané výrobcem pro platformu MS Windows nebo prostředí pod GNU licencí, které je vhodné provozovat na platformě Linux, může být také použito i pod MS Windows. Bylo využito dvou rozdílných přístupů ke kompilaci lišící se způsobem práce s make systémem.

Zpočátku jsem pracoval se systémem make souborů vytvořeným pro procesor H8S/2633. Toto prostředí jsem používal pro přípravu základní podpory pro mikroprocesor, prvotní nastavení periferií a nahrávání aplikace do mikroprocesoru. Využívání prostředí jsem ukončil přechodem na novější kompilační systém.

Nové prostředí OMK (OCERA Make System) je založeno na systému, který je používán projektem OCERA. Do tohoto prostředí byly převedeny všechny informace a nadále je využíváno jen toto.

4.1 Kompilační systém

Systém OMK pocházející z projektu OCERA viz [1] a je navržen pro použití nejen při jednoduchých projektech, ale i pro kompilaci projektů s vysokou složitostí. OMK umožňuje kompilovat různé části projektu a dává uživateli značnou volnost při tvorbě adresářové struktury.

V tomto systému byly vytvořeny základní aplikace typu „Hallo word“, aby začínající uživatel nemusel chápat a znát úlohu jednotlivých komponent a mohl tak rovnou psát vlastní program, ve kterém může využít vytvořených příkladů. Uživateli, který se nepotřebuje zabývat nahráváním aplikace, je umožněno využívat systém bez detailních znalostí kompilace a nastavení základního hardware.

4.1.1 Hlavní funkčnosti systému OMK

- centrální nastavení pravidel v souboru Makefile.rules pro většinu komponentů a jejich částí
- make systém dovoluje volně přemisťovat komponenty s křížovou závislostí, aniž by bylo třeba měnit souborové cesty
- make systém kontroluje změny v hlavičkových a zdrojových souborech a v případě změny některého z nich překompiluje a přelinkuje dotčené soubory
- je možné zavolat make jen na část projektu a tím ušetřit čas
- je možné kompilovat mimo projektový strom a tím zkompilovat z jednoho zdrojového stromu více variant najednou nebo kompilovat z adresáře určeného jen pro čtení
- tvorba make.omk souborů je uživatelsky snadná

4.1.2 Řídící proměnné

Význam proměnných použitých v souboru Makefile.omk.

`SUBDIRS` . . . seznam adresářů, které budou prohledány

`lib_LIBRARIES` . . . seznam uživatelských knihoven

`shared_LIBRARIES` . . . seznam sdílených uživatelských knihoven

`include_HEADERS` . . . seznam uživatelských hlavičkových souborů

`nobase_include_HEADERS` . . . hlavičkové soubory kopírované shodně s částí adresáře

`bin_PROGRAMS` . . . seznam požadovaných binárních programů

`utils_PROGRAMS` . . . seznam vývojových nástrojů

`xxx_SOURCES` . . . seznam cílových zdrojových souborů

`INCLUDES` . . . přidané adresáře a definice

`default_CONFIG` . . . seznam defaultních konfiguračních přiřazení CONFIG_XXX=y/n …

4.1.3 Práce s kompilačním systémem

Pokud je projekt v úvodním stavu, je třeba udělat několik kroků, než bude moci být používán.

Prvním krokem je volba cílové architektury a desky, což se provádí pomocí symbolické linky:

`ln -s` [cesta k souboru defaultní konfigurace] `config.omk`

např. `ln -s board/edk2638/config.edk2638 config.omk`.

Úvodní konfigurace se provede příkazem `make default-config` a potom lze již příkazem `make` spustit kompilaci celého projektu.

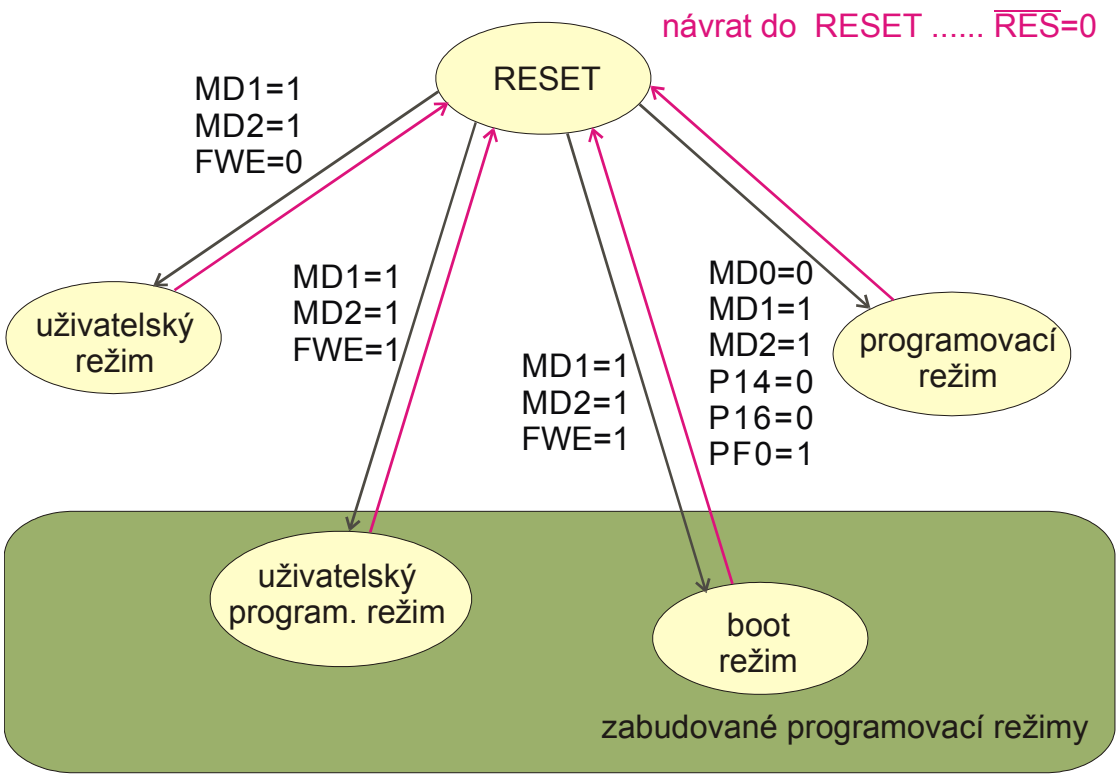
4.2 Princip nahrávání binárních souborů do zařízení

Použitím systému OMK je uživatel zcela oproštěn od potřeby znát způsob nahrávání aplikace do mikroprocesoru. Je potřeba, nastavit správně pouze režim procesoru a pak zadat příkaz v systému OMK, kterým se přenese uživatelská aplikace do mikroprocesoru.

4.2.1 Nahrávání do mikroprocesoru

Přeložený a slinkovaný projekt v podobě binárního souboru je potřeba nahrát do mikroprocesoru. To se v případě EDK2638 provádí přes sériové rozhraní. To však není obecně jediná cesta jak dostat program do zařízení. Pokud je zařízení vybaveno ještě jiným komunikačním rozhraním, lze tak někdy učinit i přes něj. Hojně je k tomu využíváno výrazně rychlejší USB.

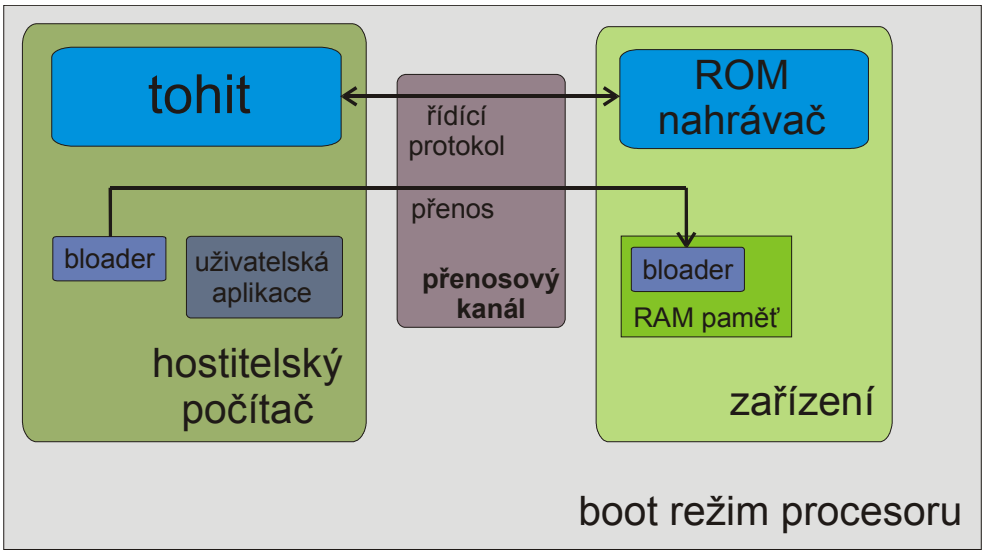
Pro nahrání aplikace jsou kromě binárního souboru a cílového zařízení potřeba nástroje k přenesení dat z PC do cílového zařízení. V tomto případě se jedná o program **bloader** přítomný v mikroprocesoru a program **tohit** přítomný v PC. Přepnutí procesoru do požadovaného režimu se provádí volbou úrovně na pinech mikroprocesoru a přivedením hodnoty 1 na pin FWE mikroprocesoru v okamžiku resetu viz Obrázek 4.1 další podrobnosti viz [3].



Obrázek 4.1: Programovací režimy.

4.2.2 Nahrávání v boot režimu

V tomto případě program **tohit** komunikuje s nahrávacím programem uloženým výrobcem v ROM mikroprocesoru a přenese program **bloader** (nebo jinou libovolnou aplikaci) do paměti RAM mikroprocesoru viz Obrázek 4.2.

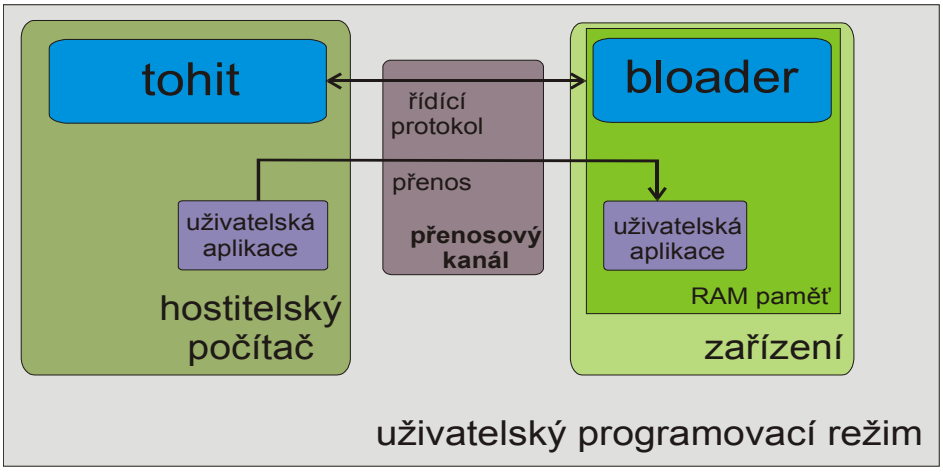


Obrázek 4.2: Nahrávání do zařízení v boot režimu.

Pokud je nahrána libovolná aplikace, je spuštěna a začne vykonávat svou činnost, když je nahrán **bloader,** je také spuštěn, ale čeká na pokyny. V tomto okamžiku je možno číst a zapisovat do paměti z PC pomocí dvojice programů **tohit** a **bloader**.

Příklad příkazu pro nahrání programu **bloader**: `sh bootstrap.`

4.2.3 Nahrávání v uživatelském programovacím režimu



Obrázek 4.3: Nahrávání do zařízení v programovacím režimu.

V tomto případě je **bloader** v paměti již přítomen, resp. je spuštěn z paměti FLASH a je možné provádět čtení a zápis do pamětí z PC.

Příklad příkazu pro nahrání uživatelského programu: `make load`.

4.3 Zavaděč v mikroprocesoru

V mikroprocesoru musí být pro nahrávání aplikace přítomný program, v tomto případě se jedná o program **bloader**. Ten je buď uložen v paměti FLASH nebo se nahraje programem **tohit** z PC do mikroprocesoru.

Tento program vykonává příkazy posílané z PC. V současné době podporuje mnoho funkcí např. čtení a zápis do paměti, spouštění programu v mikroprocesoru, příjem a posílání dat po sériovém portu.

4.4 Zavaděč v PC

Na straně PC je program **tohit**, který zajišťuje komunikaci s mikroprocesorem po sériovém kanálu. Komunikace probíhá buď pomocí protokolu, který definuje výrobce mikroprocesoru nebo odlišným protokolem vzniklým při tvorbě nástrojů pro jiný mikroprocesor.

Protokol definovaný výrobcem se využívá pokud není mikroprocesor vybaven uživatelským nahrávacím programem, který by zajistil nahrání a spuštění aplikace. V ta-

kovém případě je třeba použít režim boot, nevýhodou je, že při každém použití boot režimu dojde automaticky k úplnému vymazání obsahu FLASH paměti, které je prováděno z důvodu bezpečnosti, aby nikdo nepovolaný nemohl přečíst paměť.

Pokud je v mikroprocesoru uživatelský nahrávací program, je možné provádět operace, které jsou v nahrávacím programu implementovány.

4.5 Program tohit pro platformu MS Windows

Původní program **tohit** byl portován na platformu Windows. Tento program je určen ke komunikaci s mikroprocesorem po sériové lince z prostředí MS Windows, a jeho hlavní úlohou je nahrávání aplikací do mikrokontroléru.

Jeho vznik byl motivován snahou umožnit studentům využívat linuxové nástroje i pod Windows. Nahrání programu do mikroprocesoru vyžaduje specifickou spolupráci se sériovým portem, což tyto nástroje nezajišťují, a proto byl využit linuxový základ programu **tohit** tj. vyhodnocování parametrů a komunikační protokol, ke kterému jsem nově napsal část pracující se sériovým portem.

4.5.1 Struktura programu **tohit**

Příklad příkazů pro tohit

Příkaz pro nahrání binárního programu do procesoru:

```
/tohit --baud 57600 --sdev /dev/ttyS0 --blockmode 32 --start 0x040000 can_test-ram.bin.
```

Příkaz pro spuštění nahraného programu:

```
./tohit --baud 57600 --sdev /dev/ttyS0 --go 0x040000.
```

Popis parametrů.

- `--sdev` ... komunikační port
- `--blockmode` ... režim přenosu
- `--start` ... adresa od které jsou ukládána data
- `--go` ... adresa od které se začne vykonávat program

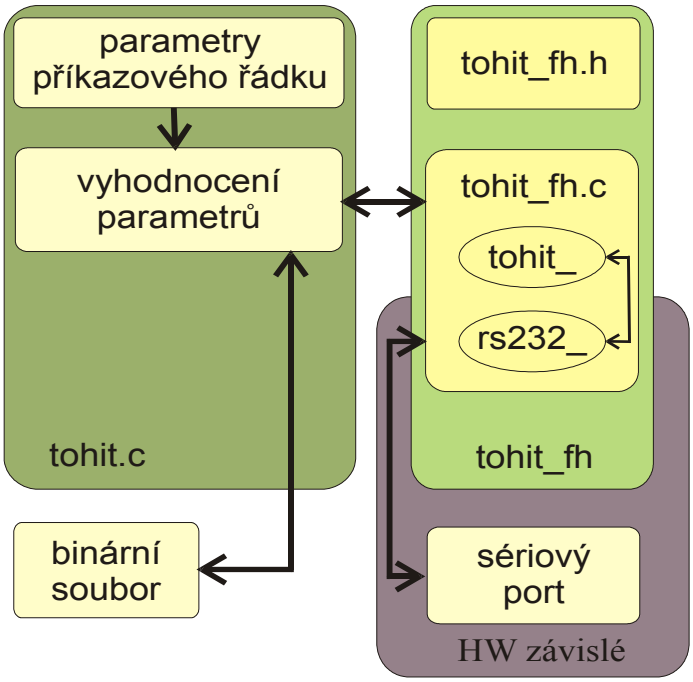
Soubor tohit.c

Obsahuje funkci pro rozpoznání parametrů zadaných při spuštění programu z příkazové řádky a pracuje se soubory pomocí standardních knihovních funkcí jazyka C.

Soubor tohit_fn.h a tohit_fn.c

Hlavičkový soubor obsahuje seznam funkcí dostupných z programu **tohit.c**.

Soubor **tohit_fn.c** obsahuje funkce dvojího druhu, funkce s předponou **tohit_** a **rs232_**. Při portaci byly funkce **tohit_** většinou zachovány, ale volání funkcí pracující se sériovým portem jsem nahradil funkcemi **rs232_**, které byly nově napsány.



Obrázek 4.4: Organizace programu **tohit**.

Funkce **tohit_** vytváří protokol pro komunikaci s programem v mikroprocesoru a realizují tak jednotlivě nebo skupinově příkazy zadané na příkazovém řádku.

Funkce **rs232_** jsou zcela HW závislé funkce pracující se sériovým portem MS Windows pomocí Windows API. Pokud by bylo třeba přizpůsobit tento program pro další systém je třeba změnit pouze tyto funkce.

5. Implementace základní podpory pro mikroprocesor

Použitím mikroprocesoru H8S/2638 vznikla potřeba přizpůsobit softwarovou podporu pro mikroprocesor s co nejlepším využitím existujících nástrojů. Důvodem byla existence vývojových nástrojů pro procesor stejné rodiny. Současně byl kladen důraz na oddělení částí lišících se pro každý mikroprocesor a závislost konfigurace na typu použité desky.

Mým cílem bylo tyto nástroje přizpůsobit pro nový mikroprocesor. Cílem celého projektu je sestavit komponenty systému a aplikací do podoby vhodné ke snadnému použití nezkušeným uživatelem nebo uživatelem, který nepotřebuje znát detaily konfigurace.

5.1 Základní podpora pro mikroprocesor

Funkcí základních nástrojů je zajistit start a běh programu. V době startu programu, po resetu mikroprocesoru, je nutné nastavit registry tak, aby byl umožněn přístup do paměti, k periferiím a nastavení přerušovacího systému. Při samotném programování se používají automaticky funkce, které pracují se zásobníkem, alokují paměť a funkce pro vstup a výstup, ale tyto funkce musí být přizpůsobeny pro danou platformu.

5.1.1 Rozdělení podle různých konfigurací

Od začátku byly základní nástroje postaveny tak, aby umožnily rozdělení na několik vzájemně odlišných částí. tj. závislosti na typu mikroprocesoru, závislost použité HW konfigurace (výběr desky) a část HW nezávislou.

Části závislé na procesoru jsou např. postupy pro nahrávání do paměti FLASH, práce s watchdogem, funkce sériových kanálů, registrace přerušovacích vektorů a práce se zásobníkem. Funkce, které vykonávají tyto činnosti jsou uloženy zvlášť v části adresářové struktury vyhrazené pro mikroprocesor. Podle zvoleného procesoru se linkují k uživatelským souborům.

Konfigurace závislá na volbě desky obsahuje např. nastavení pamětí podle připojení ke sběrnici, definici paměťových oblastí, frekvenci oscilátoru a makra pro práci s LED. Podle zvoleného typu desky jsou použity příslušné konfigurační soubory.

5.1.2 Skripty pro práci linkeru

Důležitou součástí jsou ld skripty, které definují například adresní prostory pamětí. Určují, kam se která část programu nahraje, a nebo jen definují volný prostor pro data.

5.1.3 Význam crt0

Zajišťuje start programu, překopíruje potřebné součásti programu do RAM, inicializuje RAM a volá funkci main.

5.1.4 Přizpůsobení programu **bloader**

Jedná se o port programu pro mikroprocesor ze shodné rodiny. Musel jsem upravit nastavení pro práci s některými registry, nastavení paměti a provést revizi všech prováděných činností.

Program pro svou funkci používá knihovnu **boot_fn,** která obsahuje části závislé na typu mikroprocesoru a právě ty musely být upraveny pro práci s použitým H8S/2638. Jednalo se, díky shodné rodině mikroprocesorů, hlavně o kontrolu a úpravu funkcí pracujících s pamětí FLASH.

5.2 Struktura a tvorba některých souborů

Každý programátor, který pracuje přímo s registry, už jistě řešil problém, jak vhodně pojmenovat nejen jednotlivé registry, ale i příslušné bity. Proto jsem vytvořil hlavičkový soubor pro celý procesor, aby všichni uživatelé měli shodné definice potřebných registrů.

5.2.1 Tvorba hlavičkového souboru procesoru

První krok je tvorba hlavičkového souboru, který je základním kamenem pro tvorbu programů pro konkrétní mikroprocesor.

Nejen z důvodu potřeby kompatibility již napsaných programů, jsem přejal stávající systém tvorby hlavičkového souboru. Výhodou tohoto systému je velká pře-

hlednost definic pro neobeznámené uživatele a zároveň snadné užití definic z hlavičkového souboru.

5.2.2 Struktura hlavičkového souboru procesoru

- definice přístupu k registrům
  - `__PORT8, __PORT16, __PORT32` . . . definují délku cílového registru
  - např. `#define __PORT8 (volatile __u8 * const)`
- definice ukazatele na jednotku nebo periférii
  - `#define` [jméno jednotky]_[název registru] [metoda přístupu] [adresa]
  - např. Master Control Register příslušný řadiči CAN kanálu 0 `#define HCAN0_MCR __PORT8 0xFFF800`
- definice masky registru
  - `#define` [název registru]_[název bitu resp. bitů]`m` [maska]
  - např. Master Control Register bit č.1 název bitu MCR1 maska 0x02 `#define MCR_MCR1m 0x02`
- použití v programu
  - tímto způsobem je dosaženo změny pouze u žádaného bitu
  - přiřazení 1 do Master Control Registru řadiče CANu kanálu 0 bitu MCR1 `* HCAN0_MCR |=MCR_MCR1m;`
  - přiřazení 0 do Master Control Registru řadiče CANu kanálu 0 bitu MCR1 `* HCAN0_MCR &=~MCR_MCR1m;`

5.2.3 Soubory pro konfiguraci HW

Soubor system_def_edk2638.h

Jedná se o definice pro konkrétní desku jako je frekvence oscilátoru, LED makra, definice volitelného příslušenství a jiné.

Soubor bsp0hwinit.c

Konfiguruje hardware podle desky, která je volena při vytváření počáteční konfigurace v OMK, např. nastavuje porty, adresní a datovou sběrnici a další parametry, které jsou specifické pro každou použitou desku, ale zároveň shodné pro všechny aplikace.

6. Implementace CAN knihovny

Důvodem vzniku knihovny CAN byla potřeba vytvořit vrstvu, která spojí CanFestival s HW mikroprocesoru.

6.1 Popis implementace základních CAN funkcí

V první fázi jsem implementoval funkce pro nastavení, diagnostiku, posílání a přijímání zpráv pomocí řadiče CAN. Na tomto základě jsem ověřil funkce pro příjem a posílání zpráv pro konkrétní aplikaci. Otestována a přizpůsobena byla funkce pro automatické nalezení nejlepšího nastavení CAN řadiče pro požadovanou rychlost.

Pro použití s aplikací CanFestival jsem CAN funkce začleněnil do struktury aplikace na místo HW driveru.

6.2 Organizace knihovny

Pro testování a budoucí využití slouží příkazové rozhraní, které umožňuje provádět a testovat všechny funkce knihovny.

Soubor can_test.c

Obsahuje převzatý command procesor s nově implementovanými příkazy pro práci s CAN řadičem. Jednotlivé příkazy volají potřebné funkce a tato část je tudíž HW nezávislá. Dále obsahuje struktury s nastavením CAN řadiče.

Soubor can_fn.h a can_fn.c

Hlavičkový soubor definuje funkce a struktury použité pro práci s CAN řadičem a převodní makra pro práci s registry.

Soubor **can_fn.c** obsahuje funkce pracující s HW řadiče. Jedná se o funkce pro nastavení řadiče, obsluhu přerušení, příjem a odesílání zpráv.

6.3 Popis driveru pro CanFestival

Driver zajišťuje všechny funkce potřebné pro činnost CanFestivalu, obsahuje pouze HW závislé funkce pro práci se sběrnicí CAN.

6.3.1 Inicializace CAN řadiče

Inicializace CAN řadiče se provádí zavoláním funkce

char **canInit** (*int can_chanel, long baud_rate*).

- *int can_chanel* . . . volba kanálu CAN
- *long baud_rate* . . . požadovaná rychlost v b/s
- *char canInit* . . . úspěšné provedení 0

Během inicializace se provede zapnutí napájení pro příslušný řadič, výpočet rychlosti a nastavení registrů. Po tomto volání je již řadič připraven k činnosti.

6.3.2 Výpočet nastavení pro konkrétní rychlost

Pro výpočet konkrétní rychlosti se využívá upravená funkce, která najde nejvhodnější nastavení registrů řadiče CAN pro konkrétní rychlost. Výpočet nastavení se provede funkcí

int **can_auto_baud**(*long baud_rate, int *pbrp, int *ptseg1, int *ptseg2, int sampl_pt, can_calc_const_t *c*) .

- *long baud_rate* . . . požadovaná rychlost v b/s
- *int *pbrp* . . . vrací hodnotu pro Baud Rate Prescaler
- *int *ptseg1* . . . vrací hodnotu pro Time Segment 1
- *int *ptseg2* . . . vrací hodnotu pro Time Segment 2
- *int sampl_pt* . . . pozice dělení pro Time Segment
- *can_calc_const_t *c* . . . vstupní omezení pro pbrp, ptseg1, ptseg2

Tato funkce je založena na obecném jádru, které počítá nastavení podle stanovených omezení v *can_calc_const_t *c* a vypočítá obecné hodnoty pro Baud Rate Prescaler, Time Segmenty a provede přizpůsobení obecných hodnot na hodnoty vhodné pro zápis do registrů řadiče.

6.3.3 Nastavení řídících registrů

K uložení hodnot do registrů pro nastavení rychlosti, schránek, přerušení a jiných slouží funkce

int **can_set_registers**(*int can_chanel, can_settings_t *can_settings_hcan*) .

- *int can_chanel . . .* volba kanálu CAN
- *can_settings_t *can_settings_hcan . . .* kompletní parametry řadiče

Funkce obsahuje dvě části, nastavení řadiče v režimu bitové konfigurace a přechod z režimu bitové konfigurace do normálního, v němž je nastaven zbytek potřebných registrů viz Obrázek 2.8 v kapitole 2.1.3.

6.3.4 Obsluha přerušení

Důležitou součástí driveru je obsluha přerušení, která zajišťuje potvrzení přerušení po startu řadiče při zapnutí napájení, příjem zprávy a odeslání zprávy.

Při příchodu přerušení jsou testovány jednotlivé zdroje přerušení, po nalezení zdroje přerušení je v případě příjmu či odeslání zprávy nalezena odpovídající schránka a provedena příslušná operace.

Při příchodu přerušení od resetu (zapnutí napájení jednotky) je toto přerušení smazáno a přerušení je ukončeno.

Po příchodu přerušení iniciovaném odesláním nebo příjmem zprávy je prvně nalezen zdroj tj. odeslání nebo příjem a potom odpovídající schránka.

Nalezením odpovídající schránky je v případě odeslání zprávy činnost ukončena a smazán zdroj přerušení.

Při příjmu zprávy je zavolána funkce

int **can_recive_msg**(*int can_chanel, int can_mailbox, int idx*).

- *int can_chanel . . .* volba kanálu CAN
- *int can_mailbox . . .* číslo schránky, která přijala zprávu
- *int idx . . .* parametr rezervován
- *návratová hodnota . . .* 0 při správném provedení

V této funkci je otestováno volné místo v bufferu určeném pro příjem zpráv a potom je zpráva do bufferu uložena.

6.3.5 Příjem zprávy

Příjem zprávy je složen ze dvou kroků, první je příjem z řadiče do bufferu a druhý přečtení zprávy aplikací z bufferu.

Příjem zprávy do bufferu je zajištěn během obsluhy přerušení.

Přečtení zprávy z bufferu se děje funkcí

UNS8 **f_can_receive**(*UNS8 notused, Message *msgRcv*).

- *UNS8 notused* . . . parametr nevyužit
- *Message *msgRcv* . . . přijatá zpráva
- *návratová hodnota* . . . 0x0 nepřijata žádná zpráva
  0xFF přijata nová zpráva

Tato funkce zjistí, zdali je v bufferu nová zpráva. Pokud není, provede návrat s hodnotou 0x0 a pokud je v bufferu zpráva, uloží jí do **msgRcv*, smaže z bufferu a ukončí činnost návratovou hodnotou 0xFF.

6.3.6 Odeslání zprávy

Odesílání zprávy probíhá nalezením volné schránky, pokud je k dispozici, naplněním příslušných kontrolních a datových registrů a nastavením požadavku pro odeslání v příslušném registru. Funkce odesílající zprávu je

char **canMsgTransmit**(*int can_chanel, Message msg*) .

- *int can_chanel* . . . volba kanálu CAN
- *Message msg* . . . struktura obsahující odesílanou zprávu
- *návratová hodnota* . . . 0x0 zpráva úspěšně uložena k odeslání

6.3.7 Seznam příkazů pro CAN

Příkazy se zapisují pomocí terminálu připojeného k zařízení přes sériový port.

- `CANINFO` . . . výpis nastavení registrů CAN řadiče
- `CANREC` . . . výpis přijatých zpráv

- CANSEND . . . vyslání zprávy
- CANLAFM . . . nastavení filtru příchozích zpráv
- CANNEWSET . . . nastavení definující rychlost řadiče
- CANSTART . . . aktivuje napájení řadiče
- CANSTOP . . . vypíná napájení řadiče
- CANSETREG . . . nastaví registry podle uloženého nastavení
- CANBAUD . . . automaticky nalezne nejvhodnější rychlost a nastaví
- CANSLEEP . . . přepne řadič do režimu spánku
- CANWAKE . . . probudí řadič z režimu spánku
- CANCLSTACK . . . maže zprávy z přijímacího bufferu

Příklad příkazu pro nastavení rychlosti: `CANBAUD 1 250000`, nastaví rychlost 250 kb/s na řadiči HCAN1.

7. Popis portovaného CANopen

Původně jsem uvažoval o portaci CANopen podpory a zařízení z projektu OCERA, ale rozborem jeho implementace jsem zjistil, že je výrazně rozsáhlejší než projekt CanFestival a tím převažují nevýhody složitější implementace nad funkcemi, které lze u takovéhoto malého zařízení využít, protože se nepočítá s využitím široké variability a zároveň komplexních funkcí, které nabízí. Jedná se například o dynamické vystavění slovníku podle popisu v EDS souboru za běhu aplikace.

Zvolil jsem proto projekt CanFestival-3, verze 3.0-rc1 viz [4], který je svou koncepcí vhodný k využití na tomto mikroprocesoru. Výhodou byla existující varianta pro mikroprocesor HC12, jejíž struktura souborů a část implementace nezávislá na HW byla využita.

7.1 Organizace CanFestival

Celý CanFestival lze rozdělit na tři hlavní části. HW část, vlastní implementaci CANopen standardu a uživatelskou aplikaci viz Obrázek 7.1. Součástí spadající do uživatelské aplikace je slovník zařízení (OD – Object Dictionary) obsahující údaje zařízení a uživatelské funkce. Výsledkem práce je knihovna CanFestival, která je volána z uživatelské aplikace a generické tvary callback funkcí umístěných v uživatelské aplikaci.

7.2 Význam jednotlivých souborů

Souvislosti jednotlivých souborů viz Obrázek 7.1.

Soubor canOpenDriver.c

Tento soubor tvoří společně s hlavičkovým souborem candriver.h rozhraní pro použití sběrnice CAN. Jsou v něm implementovány všechny funkce potřebné pro činnost sběrnice CAN. V hlavičkovém souboru jsou prototypy funkcí a potřebná makra pro CAN funkce.

Soubor timerhw.c

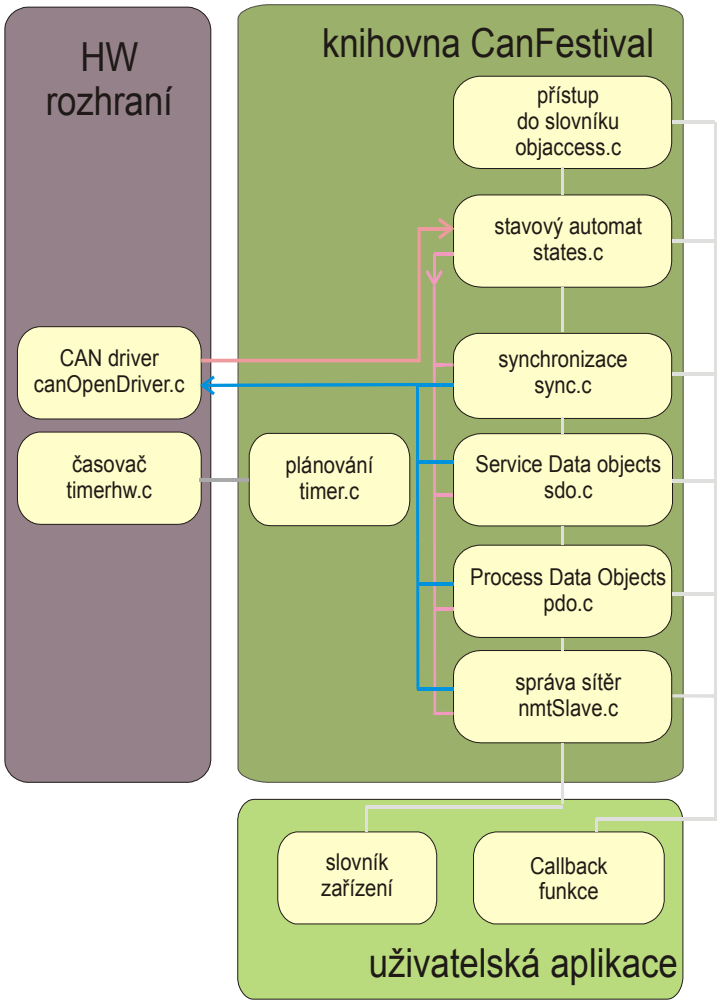
Obsahuje práci s HW časovačem, implementuje funkce pro inicializaci časovače a obsluhu jeho přerušení.

Soubor timer.c

Je implementován po vzoru časovačů v systému Linux a obsahuje funkce pro registraci časovače, zrušení časovače a pravidelná volání. Každý časovač je vybaven callback funkcí, která je volána při aktivaci časovače.

Soubor objacess.c

Soubor zajišťuje přístup do slovníku zařízení, implementuje funkce pro zápis a čtení položek slovníku.



Obrázek 7.1: Organizace knihovny CanFestival.

Soubor states.c

Jeho činností je zpracovávat došlé zprávy. Podle typu zprávy je zavolána odpovídající funkce nebo provedena změna stavu zařízení. Dále obsahuje funkce pracující s identifikací zařízení tj. čtení a nastavení čísla uzlu (*node id*).

Soubor sync.c

Stará se o synchronizaci uzlů sítě, implementuje rozesílání synchronizačních zpráv nebo jejich příjem a vyhodnocení.

Soubor sdo.c

Správu a příjem SDO zajišťuje právě tento soubor, implementuje funkce pro volání časovačů, získání a uvolnění SDO linek příjem a odesílání SDO dat.

Soubor pdo.c

Podobně jako soubor **sdo.c** zajišťuje soubor **pdo.c** správu PDO. Zajišťuje nalezení potřebných položek ve slovníku nebo naopak jejich příjem a uložení.

Soubor nmtSlave.c

Zpracovává přijaté zprávy pro správu sítě a přepíná stavy zařízení.

7.3 Implementace časovačů

V CanFestival je použita implementace časovačů odvozených od provedení v systému Linux. Je proto vhodně využit HW čítač na H8S. Časovač je puštěn v režimu volného běhu a obslužná rutina časovače nastavuje, kdy chce být aktivována podle toho, který časovač má nejkratší dobu k aktivnímu stavu.

Při obsluze je vyhodnoceno, který další časovač má být aktivní a nastaví se další čas probuzení. Tímto způsobem je dosaženo času řádu sekund. Pro další rozšíření je možné buď připojit další čítač nebo vytvořit softwarově horní bajt. Zde jsem přijal řešení se softwarovým horním bytem, který při rozlišitelnosti 55µs dosahuje přetečení za 3,6s a se softwarovým rozšířením dosahuje čekacího času 3976 minut při použití děličky frekvence hodin nastavené na 1024.

Výhodou volně běžícího časovače je možnost použít tento časovač pro další odlišné činnosti.

7.4 Práce s PDO objekty

Pro práci s časově kritickými daty slouží Process Data Objects. PDO jsou realizovány CAN zprávami s nízkým identifikačním číslem a vysokou prioritou. Mohou být posílány na základě události, v pravidelném intervalu podle synchronizačních zpráv nebo na žádost odběratele.

7.4.1 Nastavení PDO pro posílání dat

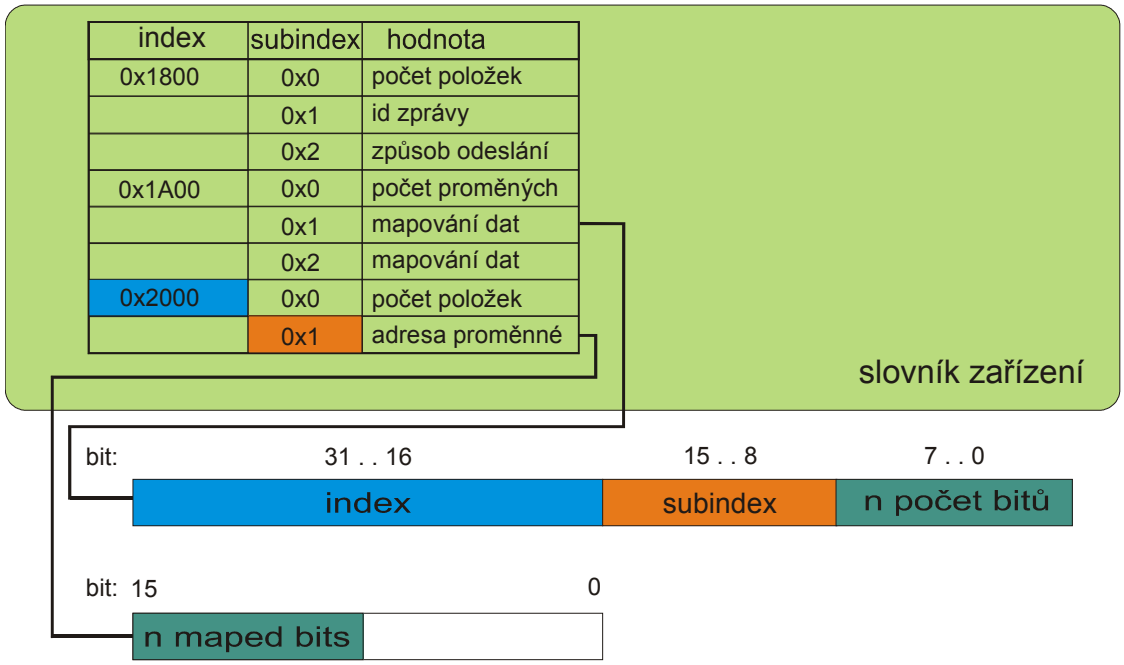
Nastavení PDO se sestává ze tří částí, definice komunikačních parametrů, určení dat, která se budou ze slovníku odesílat tj. mapování a definice dat ve slovníku.

Komunikační parametry

Toto nastavení se nachází od indexu 0x1800 a je třeba definovat dvě položky na subindexu 0x1 a 0x2. První je identifikátor PDO zprávy a druhá je metoda odesílání.

- identifikátor zprávy
  - musí se nacházet v rozsahu 0x181 – 0x57f viz [1].
- metoda odesílání
  - TRANS_EVENT, přenos vyvolaný aplikací
    - TRANS_RTR, přenos na požadavek
    - TRANS_RTR_SYNC, přenos na požadavek po příchodu synchronizační zprávy,
    - TRANS_EVERY_N_SYNC(n), přenos každou n-tou synchronizační zprávu

Mapování dat



Obrázek 7.2: Mapování proměnných do PDO.

Zde se nastavuje jaká data ze slovníku a v jaké skladbě se namapují do PDO. Tato nastavení se nachází od indexu 0x1A00, subindex 0x0 obsahuje počet mapovaných položek. Další indexy obsahují mapovací údaje.

- Definice proměnných

Proměnné situované ve slovníku zařízení se definují uložením adresy proměnné v uživatelské oblasti slovníku. Proměnné musí být definovány jinde tj. buď v samotném souboru slovníku nebo jako externí proměnná.

7.4.2 Nastavení PDO příjem dat

Provádí se shodným způsobem, ale na indexech příslušících přijímaným PDO objektům. Při příjmu se postupuje obdobně jako při vysílání dat, ale hodnota není z proměnné čtena, nýbrž do proměnné zapisována.

8. Závěr

V rámci této diplomové práce jsem se seznámil s vývojem od volby mikroprocesoru až po finální aplikaci. Poznal jsem některé obtíže, které nastávají při používání nového hardware a úspěšně je překonal, abych splnil cíle stanovené zadáním práce.

Úvodní činností mé práce bylo přepsání programu **tohit** pro použití na platformě MS Windows. Díky tomu jsem se seznámil s programem **bloader**, který je nutný pro nahrávání aplikací do mikroprocesoru.

Část, která nebyla sice přímou součástí zadání, ale tvořila nemalý podíl na mé práci, je přizpůsobení programové podpory pro mikroprocesor. Základním stavebním kamenem bylo napsání kvalitního hlavičkového souboru pro mikroprocesor. Při vytváření jsem zachoval kompatibilitu s hlavičkovým souborem mikroprocesoru shodné rodiny a umožnil jednodušší využití již hotových podpůrných programů. Další částí byla oprava programu **bloader** a nakonfigurování hardware. V této části bylo nejobtížnější nakonfigurovat vnější paměť, protože výrobce nezajistil shodné chování paměti v 8-mi a 16-ti bitovém režimu přístupu do paměti. Řešením bylo rozšířením adresní sběrnice a použití 16-ti bitového přístupu.

Pro vytvoření CANopen zařízení jsem musel implementovat driver pro práci se sběrnicí CAN. Vytvořil jsem příkazy pro existující příkazový procesor, kterými lze snadno ovládat funkce CANu přes sériový terminál, čehož jsem využil při zprovoznění komunikace s nadřízeným PC. V rámci driveru jsem též zobecnil funkci pro automatické nalezení nejlepšího nastavení rychlosti řadiče CANu a přizpůsobil tomuto zařízení.

Po vytvoření knihovny bylo třeba zvolit vhodné CANopen zařízení pro portaci. V zadání byla uvažována implementace z projektu OCERA. Při rozboru tohoto zařízení jsem dospěl k závěru, že takto rozsáhlá implementace by nebyla vhodně využita a navíc by bylo potřeba vytvořit nový způsob zpracovávání profilu zařízení vhodný pro malý procesor. Projekt CanFestival má profil zařízení určený během kompilace a jeho implementace je jednodušší, proto je pro plánované využití vhodnější. Naportována byla poslední verze 3.0 rc-1.

Projekt CanFestival jsem úspěšně začlenil do vývojového stromu OMK, spojil s CAN driverem a posléze překonal problémy vzniklé při konfliktních voláních přerušení časovačů a sériového portu.

Možnost využít vhodné real-time executivy na tomto procesoru je v současné konfiguraci těžko realizovatelné. RT exekutiva RTEMS má vyšší nároky na paměťové prostředky než je současná hardwarová konfigurace, která je k dispozici.

Využití stávající práce je díky spolupráci na projektu možné ve dvou rovinách, jednou je CAN knihovna pro další aplikace nebo kompilace konkrétního CANopen zařízení. V současné době jsem zkompiloval jednoduché zařízení posílající na sběrnici údaje o době běhu, aktuálním čase a ovládající LED na desce s možností dálkově čas nastavovat pro účely demonstrace komunikace.

9. Seznam použité literatury

[1] Open Components for Embedded Real-time Applications [online] URL: <http://www.ocera.org/archive/deliverables/ms4-month24/WP7/D7.4.pdf> [cit. 28/04/04]

[2] Ditrich Michal Komunikace mezi systémem NX5030 firmy INTRONIX [online] URL:<http://dce.felk.cvut.cz/dolezilkova/diplomky/2004/dp_2004_ditrich_michal/dp_2004_ditrich_michal.pdf> [cit. 2004]

[3] H8S/2639 H8S/2638 H8S/2636 H8S/2630 H8S/2635 Group Hardware Manual [online] URL:<http://documentation.renesas.com/eng/products/mpumcu/rej09b0103_h8s2639.pdf> [cit. FEB.22.05Rev.6.00]

[4] CanFestival [online] URL:<http://surfnet.dl.sourceforge.net/sourceforge/canfestival/CanFestival-3.0-rc1.tgz> [cit. 3.0-rc1]

[5] Evaluation Development Kit for H8S/2638 [online] URL:<http://eu.renesas.com/media/products/software_and_tools/introductory_and_evaluation_tools/european_pdfs/edk2638.pdf> [cit. 2003-03-13]

[6] HEROUT, Pavel. Učebnice jazyka C. 3. upr. vyd. České Budějovice : Kopp nakladadelství, 2001. 265 s. ISBN 80-85828-21-9

[7] CiA Draft Standard 301. CANopen application layer and communication profile: CAN in Automation, 1998

[8] CiA Draft Standard Proposal DSP-402. CANopen device profile for drives and motion control: Can in Automation, 1998