ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

Katedra řídící techniky

BAKALÁŘSKÁ PRÁCE

Rychlý vývoj aplikací v automobilové elektronice

Praha,2007 Autor:Ondřej Smítka

Prohlášení

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

V Praze dne _____ _____

podpis

Poděkování

Na tomto místě bych rád poděkoval Ing. Liborovi Wasniowskému, Ph.Dr.,vedoucímu mojí bakalářské práce , za jeho cenné rady,odbornou pomoc a snahu vyhovět mým dotazům. Můj dík patří také mé rodině a přátelům za jejich psychickou pomoc a pevné nervy.

Abstrakt

Tato práce se zabývá implementací protokolu LIN pro uzel Master. Teoretická část hlavně popisuje sběrnici LIN a komunikaci mezi uzly Master a Slave. Druhá část práce popisuje samotnou implementaci protokolu. Dalším cílem práce je demonstrovat možnosti rychlého vývoje zařízeních v automobilovém průmyslu pomocí nástroje Processor Expert pro procesory Freescale.

Abstrakt

The aim of this work is an implementation of LIN protocol for Master node. The theoretical part describes network LIN and communication between Master and Slave nodes. The second part of the work describes implementation of the protocol. The other purpos of this work is to demonstrate a possibility of a rapid prototyping in an automotive industry through the use of the tool Processor Expert for Freescale processors.

Originální zadání

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Elektronika hraje v automobilu stále důležitější roli. Snižuje průměrnou spotřebu paliva, zvyšuje výkon motoru, bezpečnost cestujících i komfort cestování, který obohacuje o multimediální zařízení a navigační systémy. Moderní polovodičové technologie přispívají ke stále rozsáhlejší náhradě původně mechanicky poháněných agregátů elektrickými. S rostoucím podílem elektroniky v automobilu se dostává do popředí problém integrace řídicích systémů a související architektura sběrnic.

S využitím systému Processor Expert je přístup k návrhu, který uživatele odstiňuje od hardware daného embedded systému, o něco jednodušší něž bylo dříve. Tento přístup je založen na použití komponent, tzv. Embedded Beans. Jelikož Processor Exper obsahuje informace o základní sadě v podporovaných procesorech, proto není nutné čekat na fyzické odlišnosti jednotlivých procesorů. Tímto způsobem je možné práci na vývoji nových zařízení značně urychlit.

Sběrnice LIN(Local Interconnect Network) je vhodná pro propojení a komunikaci smart senzorů a aktuátorů v embedded real-time systémech s rozdílnou aplikační doménou jako jsou například průmysl, automobilové systémy apod. Hlavním důvodem využití této sběrnice je hlavně nízká cena a díky využívanému UART formátu dat i možnost jednoduché implementace na libovolném mikrokontroléru.

Kapitola 2

Komunikační sběrnice LIN

2.1 Charakteristika sběrnice

LIN (Local Interconnect Network) je otevřený komunikační protokol primárně určený k propojování lokálních sítí v dopravních prostředcích (automobily, nákladní auta atd.), ale s možností využití v libovolných, pro tuto sběrnici se hodících, aplikacích například v průmyslu. Specifikace zahrnuje jak definici protokolu a fyzickou vrstvu, tak i rozhraní pro vývojové nástroje a aplikační software [1]. LIN umožňuje cenově efektivní propojení a sériovou komunikaci inteligentních (smart) senzorů a aktuátorů v embedded real-time systémech, kde není požadována velká přenosová rychlost, univerzálnost a robustnost složitějších sběrnic, například CAN.

Obrázek 1: Logo LIN sběrnice

Komunikace je založena na SCI (UART) přenosu dat, single-master/multi-slaves dialogu, jednovodičové sběrnici (nejčastěji 12V) a časovou synchronizací bez stabilizované časové základny. LIN tedy poskytuje standardní low-cost síť pro komunikaci senzorů, ale cenou

například 2krát až 3krát nižší v porovnání s CAN. V případě porovnání s CAN, by LIN by měla být spíše komplement ke CAN a ne ji plně nahradit. To je dáno odlišnými vlastnostmi obou sběrnic.

LIN protokol byl širší veřejnosti prezentován v roce 2000 konsorciem sedmi automobilových partnerů (Audi, BMW, DaimlerChrysler, Volvo, Volkwagen, Motorola a VCT), kteří začali na něm pracovat v roce 1998.

Charakteristické rysy standardní konfigurace lze shrnout do následujících bodů:

Sériový přenos dat využívající formát UART/RS-232
Jednovodičová sběrnice
Komunikace typu master-slave
Propojení až 17 jednotek (1x master, 16x slave)
Rychlost komunikace 2400 až 19200 bit/s
časová synchronizace bez stabilizované časové základny

2.2 Způsob komunikace

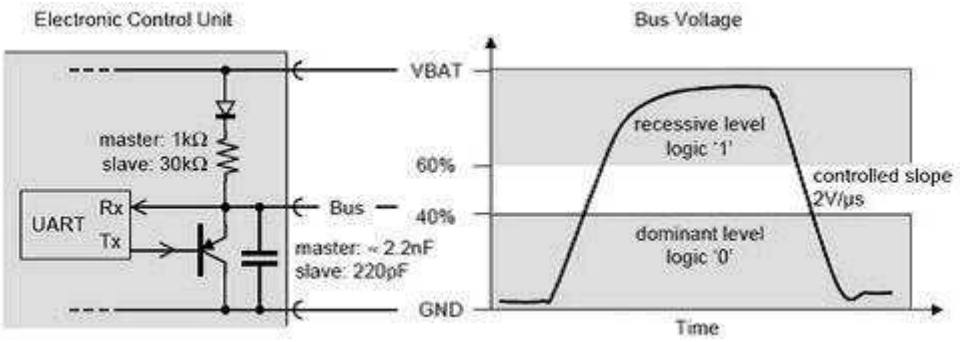
Jedná se o sběrnici typu single-master/multiple-slave, kde jedno řídící zařízení kontroluje komunikaci s jedním nebo více podřízenými zařízeními. Ke generování komunikace lze použít hardwarových a softwarových prostředků běžného UART interface (SCI), přičemž podřízené jednotky (Slave) nepotřebují k činnosti přesný krystalový generátor hodin, ale vystačí např. s RC oscilátorem. Synchronizaci pro komunikaci totiž provádí řídící zařízení (master) na začátku každé komunikace. Výše zmiňované vlastnosti mají příznivý vliv na cenu komunikačních komponent a umožňují tak snížit cenu i celých jednotlivých zařízení.

2.3 Struktura sběrnice

Konfigurace LIN sítě je složená z jedné jednotky master a jedné či několik jednotek Slave propojených sběrnicí. Ve standardu LIN 2000 je jich doporučeno použít max. 16. Master vždy iniciuje celý provoz na síti a proto zde není nutné se zabývat otázkou přístupu na síť.

2.3.1 Fyzická vrstva

LIN síť je implementována prostřednictvím pouze 1 vodiče (1-wire network), což velmi snižuje náklady na realizace, naopak však znamená vyšší hodnotu EME (Electromagnetic Emission) v porovnání se skrouceným párem (twisted pair), jak je tomu například u CAN. Z tohoto důvodu musí být zajištěná malá rychlost přeběhu a tím i menší rychlost přenosu. Pro LIN se běžně používá hodnot 2400, 9600 a 19200 bit/s.



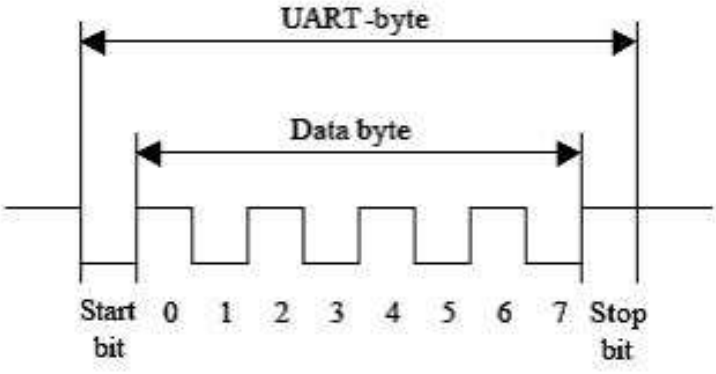
Obrázek 2: Fyzická vrstva sběrnice

Zjednodušené schéma zapojení budiče sběrnice je na obrázku 2 převzato z [6]. Vodiče VBAT a GND slouží k napájení budiče i vlastního zařízení. Pro případ přerušení dodávky napájecího napětí do zařízení připojeného na sběrnici jsou rezistory definující stav recessive, zapojeny v sérii s ochranou diodou. Ta zabrání nedefinovanému napájení jednotky po vodiči LIN sběrnice. Obecně jsou budiče sběrnice LIN konstruovány tak, aby byly odolné proti různým poruchovým stavům, které se mohou vyskytnout. Velikosti těchto rezistorů mají jmenovitou hodnotu 30 kOhm. Maximální počet zařízení připojených na sběrnici je teoreticky omezen jen počtem volných identifikátorů. Ve skutečnosti je však nutné brát zřetel na elektrické požadavky,

které počet striktně omezují. Maximální počet zařízení připojených na sběrnici by neměl překročit 16.

2.3.2 Linková vrstva

LIN protokol je vystavěn na UART (Universal Asynchronous Reciever Trasmiter) protokolu, tzn., že zprávy jsou kódovány po bajtech, což je vidět na obrázku 3 převzatého z [1]. Jedinou hlavní odlišností je tzv. synchronizační pauza (SYNC - synchronization break).



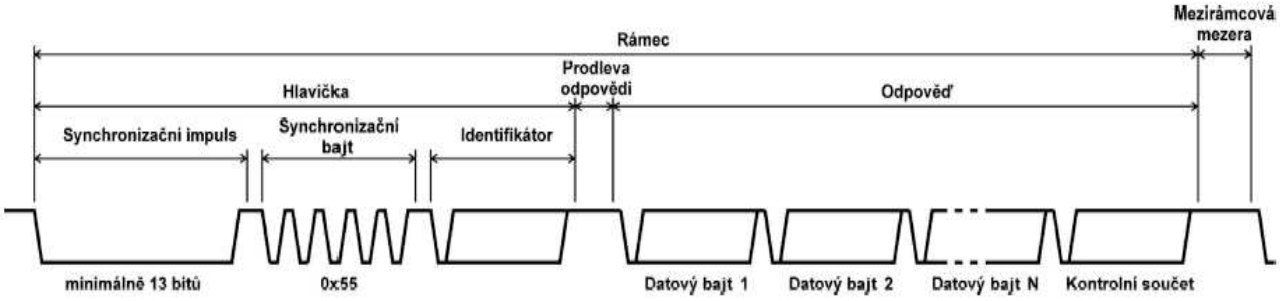
Obrázek 3: Linková vrstva sběrnice

Formát UART protokolu

LIN tedy požaduje centrální master jednotku, která komunikuje s mnoha jednotkami. Master se svou pevnou časovou základnu poskytuje datovým signálům přesnou synchronizaci se slave jednotkami a koordinuje přenos na sběrnici. Oba protokoly tedy poskytují komunikaci master-slave iniciovaný masterem. Komunikace probíhá prostřednictvím rámec LIN zprávy (LIN Message Frame).

2.4 Formát LIN zprávy

LIN používá jednotný formát rámce zprávy, který slouží k synchronizaci, adresaci uzlů a k výměně dat mezi nimi. Formát rámce zprávy je na obrázku 4 převzatého z [6]. Řídící jednotka (master) začíná komunikaci, určuje přenosovou rychlost a vysílá hlavičku rámce zprávy. Ostatní jednotky, ale i jednotka master mohou vysílat odpověď složenou z datových bajtů a kontrolního součtu. Hlavička začíná synchronizačním impulsem a následným synchronizačním polem. Toto pole slouží k zasynchronizování podřízených jednotek (slaves) na bitovou rychlost jednotky master. Tyto jednotky tak vystačí s jednoduchým zdrojem časové základny v podobě RC oscilátoru, což má kladný vliv na cenu jednotek slaves.



Obrázek 4: Rámec zprávy

2.4.1 Header frame(Hlavička)

Header frame, resp. Command frame má vždy konstantní pevnou délku a je rozdělen na 3 části (dle Tabulky 1):

SYNC-break synchronizační pauza	SYNC-field synchronizační pole	ID – identifikátor	
1 B	1 B	6b-adresové pole	1b-paritní pole

Tabulka 1:Formát Hlavičky zprávy

Synchronizační pauza (Synchronization break)

SYNC-break v první části LIN zprávy je složen z minimálně 13 bitů nul. Tato pauza je určena k tomu, aby slaves jednotky spolehlivě detekovaly zprávy vyslané na sběrnici.

Když je SYNC-break přijata, LIN softwarová rutina ve slave jednotce musí zkontrolovat, zda všechny následně přijaté bity jsou nuly, aby bylo zajištěno, že pauza byla detekována. Pro master jednotku, který může být implementována na mikrokontroléru s UART, může být problém vytvořit pomocí UART sled 13 bitů nul. Jednou cestou, jak toho dosáhnout, je při generování pauzy snížit baud rate UART vysílače masteru a tak čas potřebný na vyslání 9 bitů v UART je ve slave jednotkách interpretováno jako 13 bitů nul ve správné baud rate.

Synchronizační pole (Synchronization field)

Master inicializuje přenos vysláním header framu, podle které jsou jednotky slave schopny synchronizovat svoje hodiny vždy předtím, než je přijatá nová zpráva. To umožňuje neimplementaci drahého a citlivého rezonátoru nebo oscilátoru v každém slave uzlu a pouze jeden přesný rezonátor je vyžadován pro master jako časová reference. Synchronizace slave jednotek je tedy dána odměřením času od první sestupné hrany (start bit) k páté sestupné hraně (tj 7.bit) synchronizačního bajtu (SYNC-byte) a vydělenou 8, aby byla zjištěna baud rate masteru.

Identifikační pole (Identifier field)

Poslední část zprávy hlavičky je ID-pole. Toto pole označuje následně vysílaný datový rámec (data frame) a je chráněno dvoubitovou paritou. V tomto poli jsou dva bity určené ke specifikaci délky datového rámce (2, 4 nebo 8 bajtů). V nejnovější verzi však je definována i hodnota 0bajtů. Slave uzel je v síti adresován ID-polem. Uzly však nemají přiřazeny fyzické adresy jako je například MAC adresa. Místo toho je předdefinovaný seznam platných ID.

2.4.2 Data frame (Datový rámec)

Datový rámec (data frame), resp. rámec odpovědi (response frame) je složen z 0 až 8 bajtů. Délka tedy může být různá dle hardwarové a softwarové implementaci každého uzlu. Na je kontrolní součet, který je počítán z vysílaných dat.

0 až 8 B - variabilní délka datového pole	1B - check byte

Tabulka 2:Formát Datového rámce

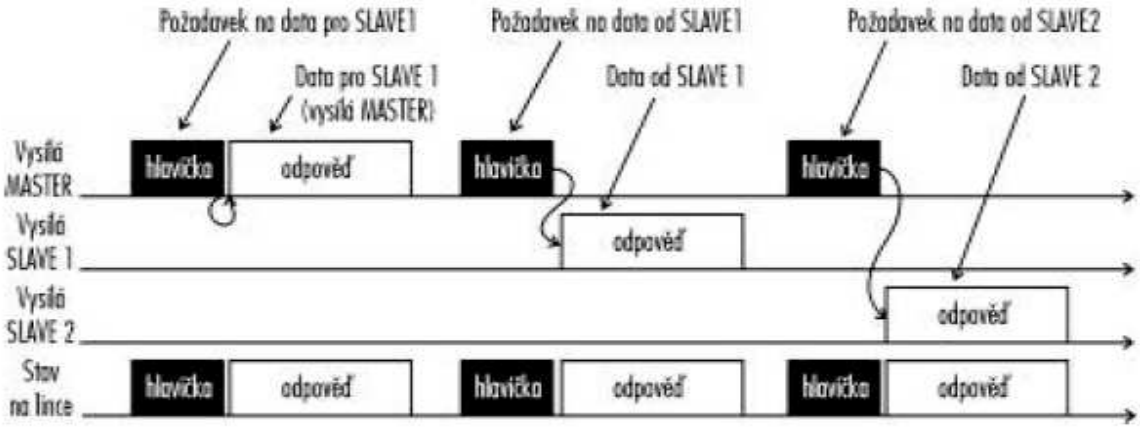
2.5 Zasílání dat na sběrnici mezi Mastrem a slave

Master -> Slave

Master vyšle oba rámce, tedy hlavičku i datový rámec, jedné či více jednotkám slave (multicast, broadcast).

Slave -> Master

Tento směr komunikace nastane v případě, že master vyžaduje odpověď od určité slave jednotky. Příklad průběhu komunikace po sběrnici je vidět na obrázku 5 převzatého z [6] .



Obrázek 5: Komunikace na sběrnici

2.6 Sleep mod,WakeUP signál

Sleep mód

Tento mód je vhodný v případě, že na sběrnici dochází jen ke sporadické komunikaci a tedy v dlouhých pasážích bez komunikace není nutná plná funkce. Master aktivuje sleep mód prostřednictvím řídícího rámce s ID = 0x3C s prvním data bajtem rovným 0x00 vyslaného všem jednotkám (broadcast). Slave jednotky se po tomto příkazu přepnou do nízkopříkonového (low-power) módu.Přechod do Sleep módu je možný po uplynutí doby TTIME_OUT. Přepnutí do normálního režimu je možné jedině WakeUp signálem vyslaným na sběrnici jednotkou Master nebo Slave.

WakeUP mód

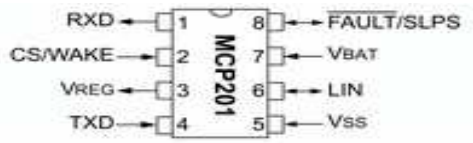
Jedinou možností jak probudit jednotky ze Sleep modu je vyslání na sběrnici WakeUp signál 0x80.Pokud není komunikační rychlost mezi Mastrem a Slavem synchronizvána můžou se detekovat znaky 0xC0,0x80,0x00,což je 7 až 9 dominantních bitů. Dále WakeUP obsahuje minimálně 4recesivní bity,jako oddělovač před vysláním Synch Break .Pokud od vyslání WakeUp signálu do doby TT0BRK nevyšle Master Synch Break může Slave opakovat vysílání WakeUp signálu třikrát.Pokud stále Master nereaguje další vysílání může nastat až po uplynutí doby TT3BRK.

2.7 Praktická realizace

Převodník MCP201

Integrovaný převodník MCP201 firmy Microchip [4] poskytuje fyzické rozhraní mezi poloduplexní sběrnicí LIN a vstupy/výstupy mikrokontrolérů a mikroprocesorů. Konkrétně na

pinu LIN poskytuje poloduplexní obousměrný přenos dat standardem LIN specifikace 1.3 a naopak na pinech TXD/RXD nabízí klasické TTL/CMOS logické rozhraní dle standardu UART (Universal Asynchronous Receiver/Transmiter). Dále na pinu VREG poskytuje pro další obvody napájecí napětí 5 V/ 50 mA z napájecího 6 až 18 V. Chip také obsahuje tepelnou ochranu obvodu, která v případě překročení kritické úrovně vypne LIN vysílač a vnitřní napěťový 5 V stabilizátor.



Obrázek 6: MCP 201

Hlavní parametry obvodu:

Napájení (pin VBAT): 6.0V až 18.0V (max. trvalé nedestruktivní 30 V)

Spotřeba: max. 1 mA (max. 50 µA v Power-down módu)

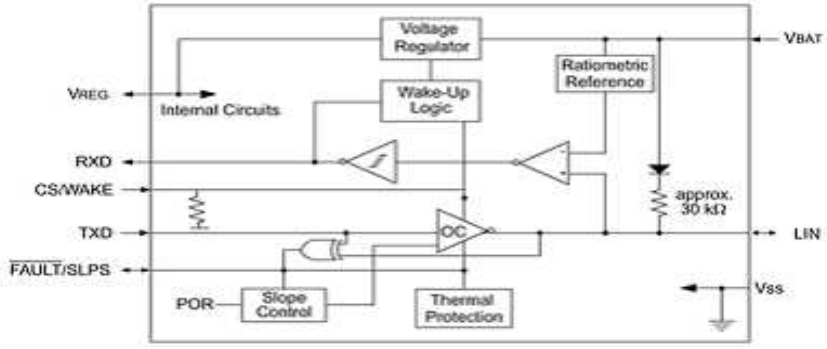
Přenosová rychlost: 1 až 20 kb/s

Rozhraní sběrnice LIN: LIN Protocol Specifikace 1.3., rychlost přeběhu budiče volitelná pinem SLPS mezi typ. 2 a 4 V / µs, úrovně:do 0.2 VBAT (log. 0) a nad 0.8 VBAT (log. 1)

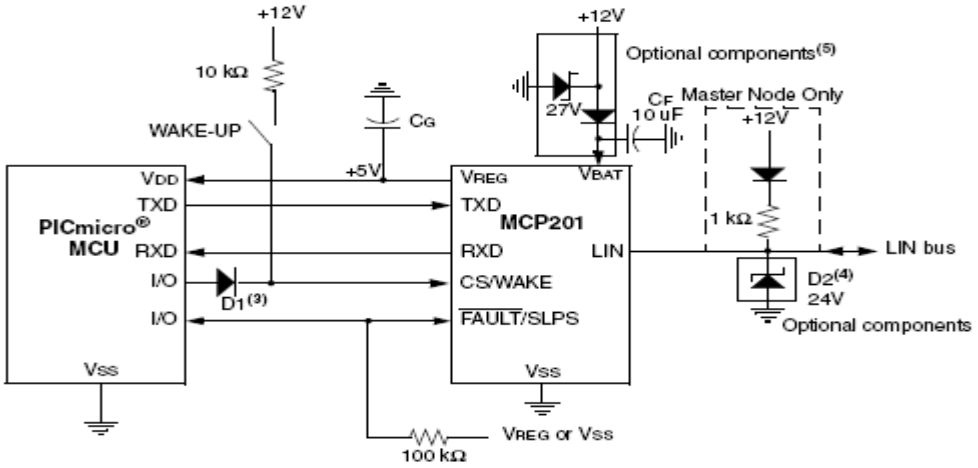
Rozhraní sériové TX, RX: úrovně kompatibilní s 5V CMOS/TTL I/O terminály (TX: do 0.15 VREG log.0 a nad 2V log. 1, RX: do 0.2 VREG log.0 a nad 0.8 VREG log. 1)

Pracovní teplota: -40°C až +125°C, automatický teplotní shutdown

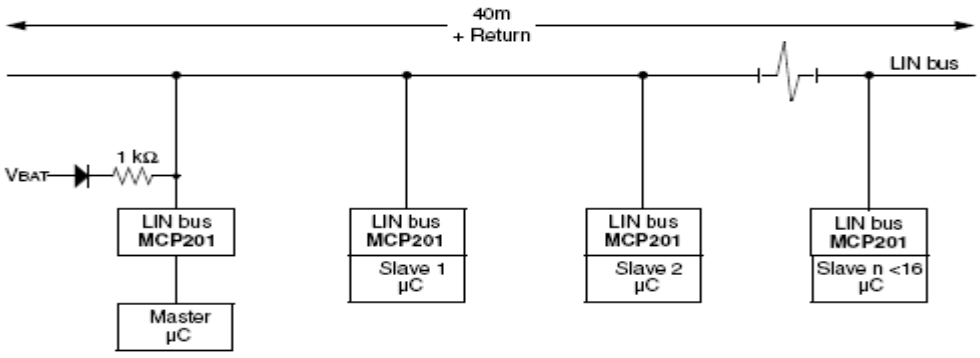
Další: tepelná a ESD ochrana, ochrana přepólování a krátkodobého přepětí



Obrázek 7: Vnitřní struktura MCP 201



Obrázek 8: Typická aplikace s MCP 201



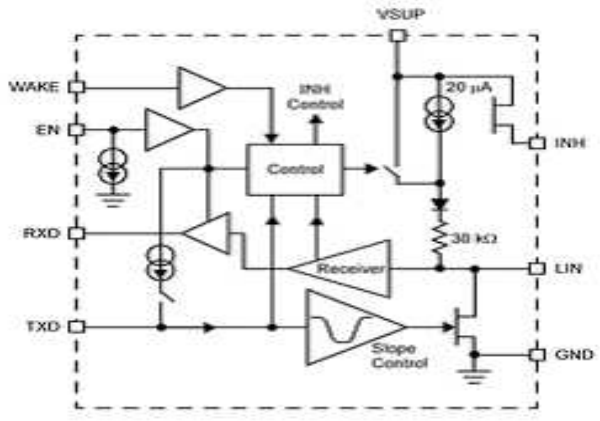
Obrázek 9: Typická zapojení na sběrnici

Převodník MC33661

Převodník Freescale MC33661 nabízí proti obvodu MC33399 rozšířené LIN rozhraní specifikace 2.0 optimalizované pro minimální elektromagnetické vyzařování [3].Toho je dosaženo použitím různé rychlosti přeběhu pro různé přenosové rychlosti. V tzv. fast módu pak může být prováděn přenos rychlosti přes 100 kb/s díky rychlosti přeběhu 15 V/µs. Dalším velkým rozdílem je podpora 5V i 3.3 V úrovně logických signálů na straně klasického sériového rozhraní kompatibilního s CMOS i TTL.

Hlavní parametry obvodu:

- Napájení (VNAP): 7.0V až 27.0V (max. 40 V)
- Spotřeba: max. 8 mA (typ. 8 µA, max. 200 µA v sleep módu)
- Přenosová rychlost: 1 až 20 kb/s (normal mód), 1 až 10 kb/s (slow mód - poloviční rychlost přeběhu SR), 100 kb/s (fast mód - vysoká rychlost přeběhu SR)
- Rozhraní sběrnice LIN: LIN Protocol Specifikace 2.0, nastavitelná SR pro optimalizaci vyzařování (max. ve fast módu 15 V/µs), úrovně: do 0.4 VNAP (log. 0) a nad 0.6 VNAP (log. 1)
- Rozhraní sériové TX, RX: kompatibilní s 5 a 3.3 V logikou CMOS/TTL (TX: do 1.2 V log.0 a nad 2.5 log.1, RX: 0 - 0.9 V log. 0 a 4.25 - 5.25 V log.1 (5 V logika)
- Zpoždění TX,RX <-> LIN: max. 6 µs
- Pracovní teplota: -40°C až +125°C, automatický teplotní shutdown
- Další: nadproudová ochrana sběrnice, tepelná ochrana, detekce podpětí, sleep mód/probuzení, enable vstup



Obrázek 10: Vnitřní struktura MC33661

Další výrobci LIN budičů

Dále pro orientaci uvádím některé typy převodníků, které jsem našel včetně odkazů na webové stránky výrobců.

– Atmel (http://www.atmel-wm.com) – ATA6622, ATA6624, ATA6661, ATA6662
– Freescale (http://www.freescale.com/) – MC33689, MC33661
– Philips (http://www.semiconductors.philips.com/automotive) – TJA1020U/N1
– STMicroelectronics (http://eu.st.com) – L9638

Kapitola 3

MC56F8367EVM

3.1 Základní popis

MC56F8367EVM je užívaný k demonstraci schopností 56F8367 hybridních procesorů a poskytuje hardwarové nástroje pro následné vývojové využití [3].

MC56F8367EVM je výpočetní deska, která zahrnuje procesor 56F8367, periferní rozšiřující konektory, CAN interface, 512KB externí paměti a dvojici konektorů pro monitoring signálů a budoucí uživatelské rozšíření.

MC56F8367EVM je konstruována pro následující účely:

Poskytnutí novým uživatelům seznámení s výhodami architektury 56800E.Také nástroje a příklady poskytnutý s MC56F8367EVM napomáhají k rychlému nasazení do konkrétní aplikace.

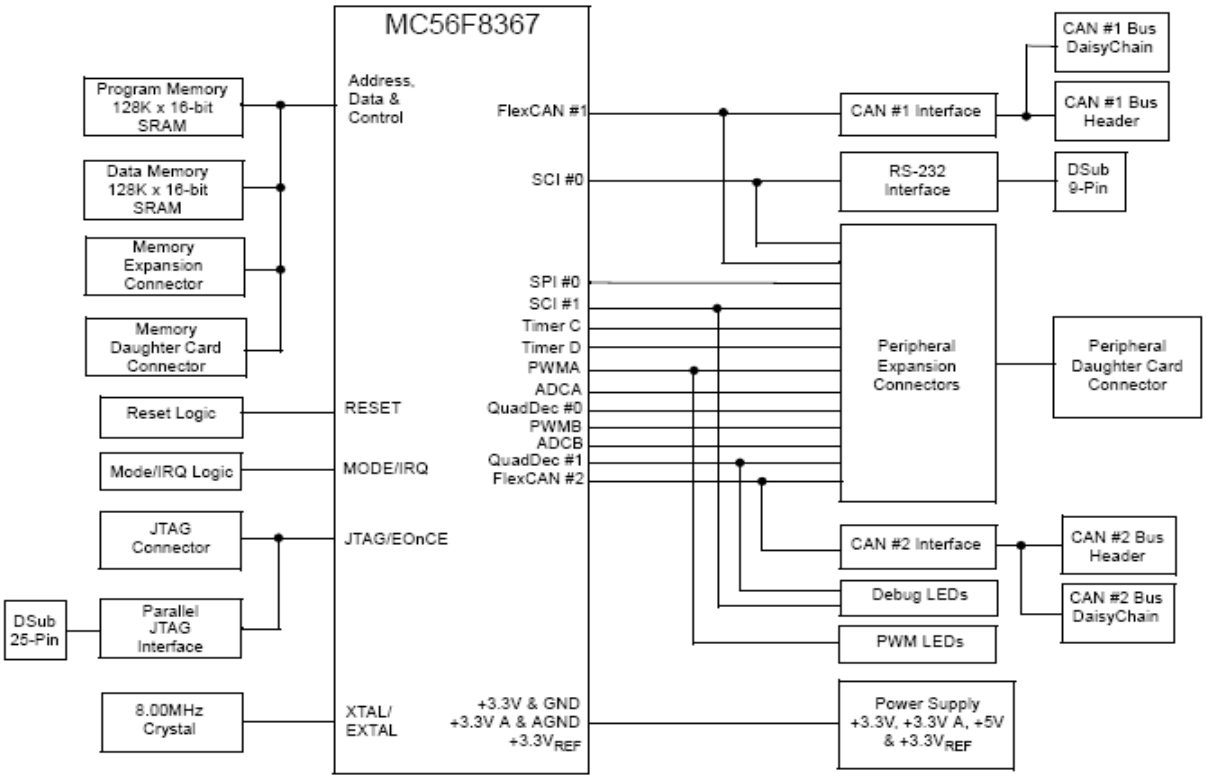
Složí jako platforma pro vývoj aplikací v reálném čase. Tato nástrojová sada vhodná pro uživatele k vývoji a běžné simulaci , nahrání programu na chip nebo do RAM paměti,běh programu a lazení užívá JTAG/Enhanced OnCE (EOnCE) port.Breakpoint se nastaví přes EOnCE port uživatel snadno specifikuje místa,kde se má program zastavit. Schopnost prohlížet a upravovat všechny přístupné registry, paměti a periferie přes EOnCE port značně ulehčuje práci pro programátora.

Slouží jako platforma pro hardwarový vývoj.Hadwarová platforma umožňuje uživateli propojení s externím technickým vybavením. Na desce mohou být periferie deaktivovány to umožňuje uživateli připojení žádné až všech procesorových periferií.

3.2 Architektura MC56F8367EVM

MC56F8367EVM může být užíván k vývoji v reálném čase programových a hardwarových produktů.

MC56F8367EVM poskytuje vlastnosti nutný pro uživatele při psaní a ladění programové vybavení,demonstraci funkce tohoto programového vybavení a periferií přímo v uživatelské aplikaci.MC56F8367EVM je dost flexibilní a povoluje uživateli plně využít výhody 56F8367 a optimalizuje výkon jejich produktů. Architektura MC56F3867 je vidět na obrázku10 převzatého z [3] .



Obrázek 11: Architektura MC56F3867

3.3 Technická specifikace

MC56F8367EVM je konstruovaná jako všestranná vývojová deska s procesorem 56F8367,umožňující vytvořit programové a hardwarové produkty v reálném čase.Podpora nové generace aplikací jako je řízení motorů a servo pohonů, digitální a bezdrátová komunikace, digitálně odpovídající systémy, telefony, modemy a digitální kamery. Výkon 16- bitových procesorů 56F8367 v kombinaci s 128K x 16- bit externí programovou/datovou statickou pamětí RAM (SRAM), 128K x 16- bit externí data/program SRAM, RS - 232 port, CAN port, periferní konektory a paralelní JTAG konektor dělá MC56F8367EVM ideální pro vývoj a implementaci mnoho řídících algoritmů pro motory a také jak pro učení architektury a instrukční soupravy procesoru 56F8367.

Hlavní přednosti MC56F8367EVM:

- MC56F8367VPY60, 16-bit ,+3.3V/+2.5V hybridní procesor na frekvenci 60Mhz
- Externí rychlá RAM (FSRAM) paměť 128K x 16-bit
- 8.00MHz krystalový oscilátor pro základní frekvenci
- Komunikační porty RS-232,CAN, FlexCAN, SPI,SCI
- Pulzně šířková modulace PWMA a PWMB
- ADC převodníky,časovače a další
- Další perifetie tlačítky,LED diody,teplotní senzor a mnohé další

3.4 CodeWarrior

CodeWarrior Development Studio od firmy Metrowerks je integrované vývojové prostředí s grafickým uživatelským rozhraním určené pro vývoj embedded aplikací,jehož součástí je simulátor instrukční sady pro procesory od firmy Motorola.

Toto IDE dále obsahuje nástroj pro správu projektů, nástroj pro verzování,optimalizující C a C++ kompilátor a linker, debugger na úrovni zdrojového kódu, editor pro vytváření a úpravu souborů se zdrojovým kódem a jiných textových souborů, vyhledávací nástroj umožňující vyhledávání a nahrazování řetězců i regulárních výrazů v textech a porovnávání souborů,

zdrojový prohlížeč, který si udržuje databázi symbolů, a usnadňuje tak orientaci v kódu při vyhledávání a debuggování.

CodeWarrior Development Studio je multiplatformní aplikace, která je konzistentní přes všechny podporované pracovní stanice a PC – na všech platformách jsou její funkce a vlastnosti stejné.

Grafické uživatelské rozhraní je založeno na standardních widgetech a jako základní prvek slouží okna. Všechny nástroje, které jsou součástí tohoto IDE využívají GUI a jsou po stránce vzhledu i funkčnosti do velké míry konfigurovatelná uživatelem.

Debugger, který je součástí IDE, poskytuje celou řadu funkcí, z nichž většina je přístupná přes intuitivní point-and-click rozhraní. Tento nástroj umožňuje vytváření uživatelem definovaných oken, tlačítek a dalších prvků GUI a poskytuje jak vysokoúrovňové pohled na software, tak nízkoúrovňové zobrazení hardwaru – tyto pohledy jsou dále uživatelsky konfigurovatelné, je například možné volit si formát zobrazovaných dat.

Z hlediska vlastní funkčnosti umožňuje debuggovací nástroj snadné nastavení a zobrazení nejen breakpointů, ale i tzv. eventpointů. Eventpointy se liší od breakpointů tím,že jejich činnost nespočívá v zastavení simulace při splnění jejich podmínky, ale v provedení jiné činnosti – zápis do logovacího souboru, výpis na obrazovku, spuštění trasování, skriptu,změna toku programu atp. Jedním z typů eventpointu je i tzv. watchpoint – jde o podmíněný nebo nepodmíněný breakpoint na určitém místě v paměti, který zastaví simulaci při přístupu nebo změně daného paměťového místa.

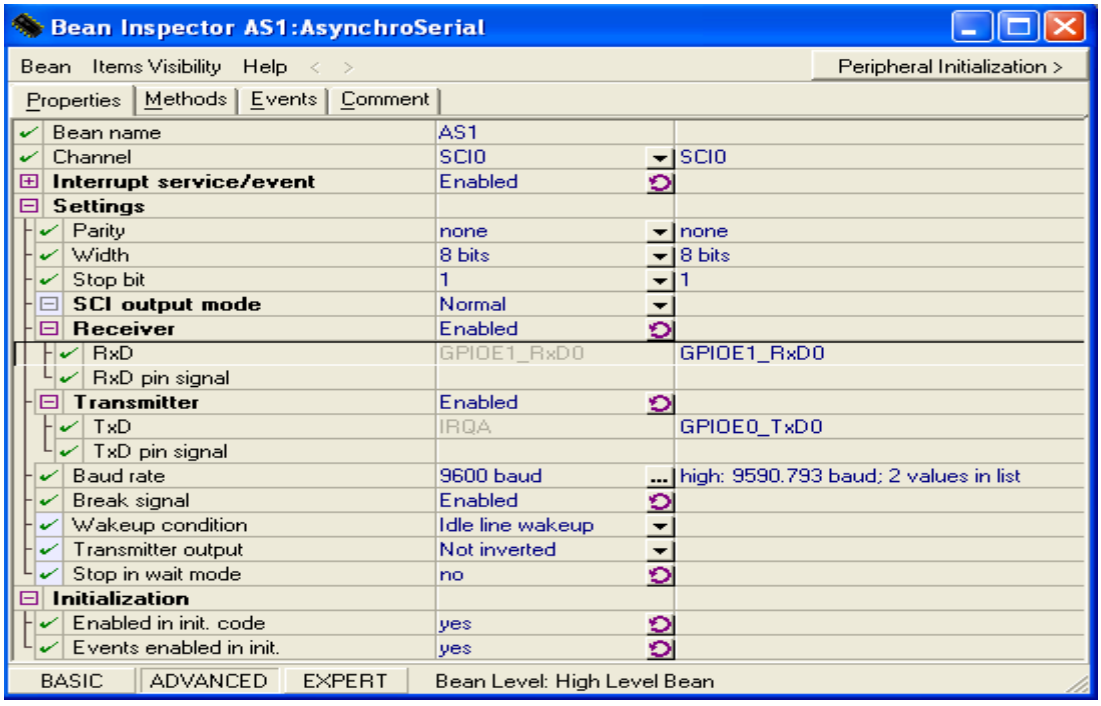
Další podporovanou funkcí je krokování (step into, step over, step out). Protože se jedná o debugger, který nepracuje jen na úrovni strojového kódu, ale i vyššího jazyka,umožňuje vypisování call stacku funkcí, sledování hodnot proměnných, zobrazování kódu v nižší, vyšší nebo „mixované“ podobě, zastavení simulace při detekování vzniku výjimky jazyka C++, atp. Další z funkcí debuggeru je debuggování multi-procesorových aplikací.

3.5 Processor Expert

Processor Expert (PE) je vývojové prostředí [2] podporující mikrokontroléry Freescale určené pro rychlý vývoj aplikací s využitím programovacího jazyka C. PE má k dispozici informace o procesorech a jejich periferiích,proto je možmé provádět nastavení i pokud není ještě k fyzicky k dispozici samotný procesor.

Tyto informace o jádře procesoru a jeho periferiích jsou obsaženy v tzv. Embedded Beans. Tyto speciální beany jsou již vytvořeny a postupně se vylepšují,vytvářejí nové nebo vznikají nové jako nadstavba na již vytvořené. Podle těchto beanů PE generuje kód s využitím nastavení hardwarových vlastnosí procesoru.

Nastavení jednotlivý beanů je velmi jednoduché díky grafickému uživatelskému rozhraní. Bean je tvořen vlastnostmi (properties), metodami (methods) a událostmi (events), což je vidět na obrázku 11 .Dle specifikace podle zvoleného procesoru PE kontroluje nastavení.Pokud není nastavení korektní, tak bean vyhodnotí chybu a omezí jeho používání.

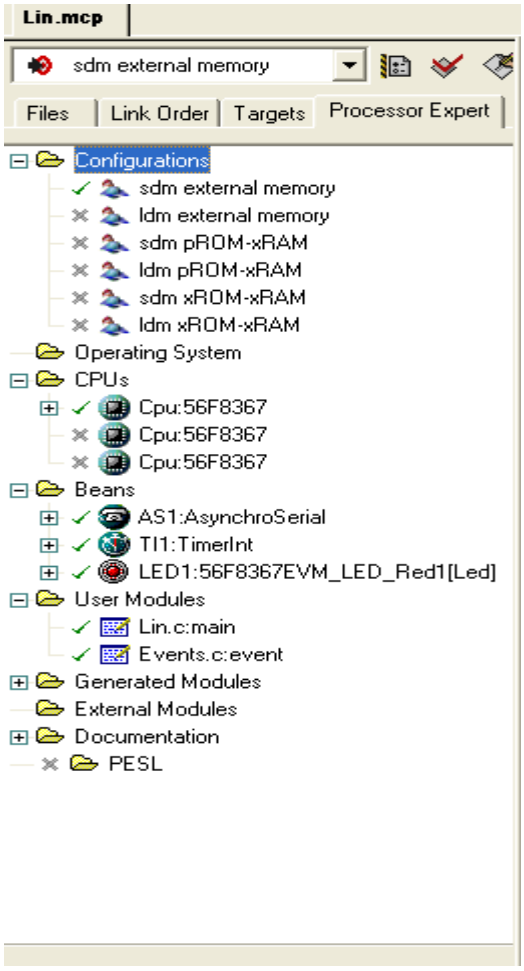


Obrázek 12: Bean Inspector

Pomocí vlastností se nastaví parametry hadware bez konkrétní znalosti detailů procesoru a jeho registrů.Všechna nastavení se nechají kdykoliv měnit v případě změny zapojení výstupních periferií.Metody umožňují uživatelský přístup k periferiím.Tyto metody se nechají vkládat přímo do uživatelského kódu.Události se využívají k ošetření přerušení od jednotlivých periferií.

Všechny popsané výhody pomáhají uživatelům vytvářet rychle a kvalitně program v kratší době a za nižší cenu.

Vytvoření projektu



Obrázek 13: Processor Expert

Při vytvoření projektu je možnost vybrat prázdný nebo již hotový projekt,který slouží jako příklad nebo předloha pro uživatelskou tvorbu

V záložce CPU je možno změnit typ procesoru

Do záložky Beans se vkládají jednotlivé Beany,které budeme potřebovat.V jednotlivých Beanech se skrývají konkrétní metody,ktré se nechají vkládat do uživatelského kódu pomocí funkce drag&drop .

User Modules(uživatelské moduly) obsahují vlastní uživatelské kódy

Generated Modules(generované moduly) obsahují zdrojové kódy pro jednotlivé Beany v záložce Bean Modules a ostatní kódy pro další potřebné moduly

V tomto menu je možné pohodlně měnit parametry jednotlivých Beanů viz. obrázek 12

Kapitola 4

Analyzátor LIN

4.1 CANcaseXL

CANcaseXL je zařízení od firmy Vector [8] na obrázku 13, které měří, zaznamenává, simuluje, testuje mobilní a stacionární sítě. Jde o robustní měřící modul s rozhraním USB pro práci se sběrnicí CAN/LIN. CANcaseXL se hlavně využívá pro záznam dat ze sběrnice. Díky robustnímu provedení je možné zařízení provozovat v náročných podmínkách.Toto zařízení je universální, protože podle použití se nechají připojovat různé moduly pro konkrétní typ sběrnice.Pro sběrnici LIN je CANcaseXL osazen modulem LINpiggy 6259opto.

Obrázek 14: Analyzátor CANcaseXL

4.1.1 Vlastnosti

Je vybavený pamět'ovou deskou pro připojení SD karty nebo připojen do PC.Jelikož je zařízení možné provozovat bez PC jen s SD kartou,na kterou je možné nashromážďit až 2GB dat .Tyto data se potom nechají anylyzovat v počítači pomocí programů CANalyzer nebo CANgraph.

4.1.2 Funkce

Umí vysílat a přijímat data ze sběrnice bez narušení chodu sběrnice. Pro sběrnici LIN je analyzátor možné nastavit jako jednotka Master nebo Slave nebo pouze monitoruje sběrnici.V samotném programu CANanalyzer je možné s daty pracovat jako např. přijímat, vysílat nebo opravovat chybné zprávy. Těmito vlastnostmi a nástroji se nechá sběrnice dobře otestovat jako kdyby byla umístěna v náročných podmínkách. Jelikož CANcaseXL v konfiguraci s LINpiggy 6259opto má dva samostatné kanály je možné simulovat například Master i Slave jednotku.

4.2 LINpiggy6259opto

Jde o přídavný modul do CANcaseXL.V tomto případě jde o modul pro sběrnici LIN s galvanickým oddělením,které chrání modul před poškozením při poruše sběrnice, tento modul je vidět na obrázku 14 převzatý z [8].Tyto moduly se vyrábějí i pro sběrnici CAN, pak je možné do CANcaseXL umístit jeden modul pro LIN a druhý pro CAN a potom analyzovat data ze dvou sběrnic navzájem.Tato funkce je podporována i softwarem od firmy Vector.

Obrázek 15: LINpiggy6259opto

4.3 Instalace CANcaseXL

Instalace připojeného modulu se provádí pomocí CD CANcaseXL nebo jsou drivery ke stažení z [8] . Hardwarovou část je nutné udělat manualně, protože CD neobsahuje žádný instalační wizard, ale pouhé drivery. Po instalaci driverů je nutné nainstalovat software pro snímání dat z analyzátoru, tento software je na dalšímm přiloženém CD CANalyzer/DENalyzer nebo ke stažení nejnovější verze z [8]. Výchozí nastavení nainstaluje veškerý software pro práci se všemi zařízeními od firmy Vector. Pro sběr dat a jejich analýzu se používá nástroj CANanalyzer, na CD v licencované verzi lite. Před prvním užitím programu je nutné nastavit typ analyzátorů a kanály na kterých jsou připojeny. To se provádí pomocí utillity VectorHardware, která přibude v ovládacích panelech po instalaci driveru.

Kapitola 5

Vlastní práce

5.1 Programování jednotky MASTER na LIN

Instalace

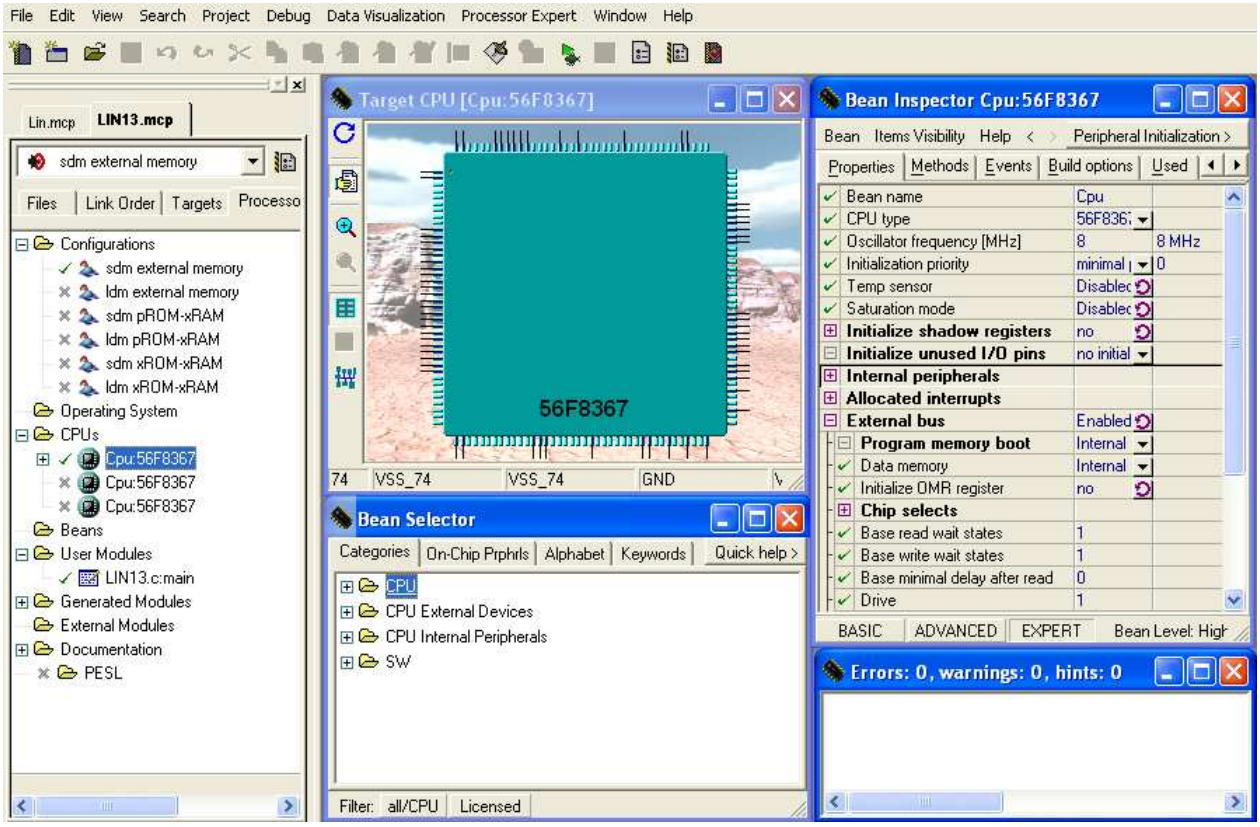
Na začátku programování jednotky bylo potřeba nainstalovat potřebný software pro samotnou práci. Nejdříve jsem na počítač nainstaloval program CodeWarrior,který se aktualizoval CW 7.1 UPDATE pro řadu DSP56800.Potom se přidal plugin pro Procesor Expert . Nakonec už jenom zbývalo nahrát licenci license.dat pro CodeWarrior do kořenového adresáře c:\Program Files \Metrowerks \CodeWarrior .

Seznámení s vývojovým prostředím

Na začátku práce jsem si nejdříve spustil pár demo příkladů podle kterých jsem se seznámil s vlastní tvorbou programu a funkcemi vývojového prostředí.Díky těmto příkladům nebylo těžké vytvořit svoji první aplikaci s LED diodou a seriovou komunikaci mezi počítačem a MC56F8367EVM .V této části jsem si hlavně vyzkoušel práci s nastavením vlastností procesoru, jednotlivých Beanů a způsobu programování.

Vytvoření nového projektu a nastavení beanů

Vytvoření nového projektu File/New/ProcesorExpert Stationery se vyplní název projektu Lin.Následně již stačí zvolit typ procesoru a projekt je vytvořen, což je vidět na obrázku 15.



Obrázek 16: Vytvoření nového projektu

V levé části okna se nachází nástroj Processor Expert popsaný v kapitole 3.5

V okně Target CPU je zobrazen vlastní procesor s obsazeností vývodů.Tento pohled je možné přepnout do režimu blokového zobrazení využitelnosti jednotlivých portů procesoru.

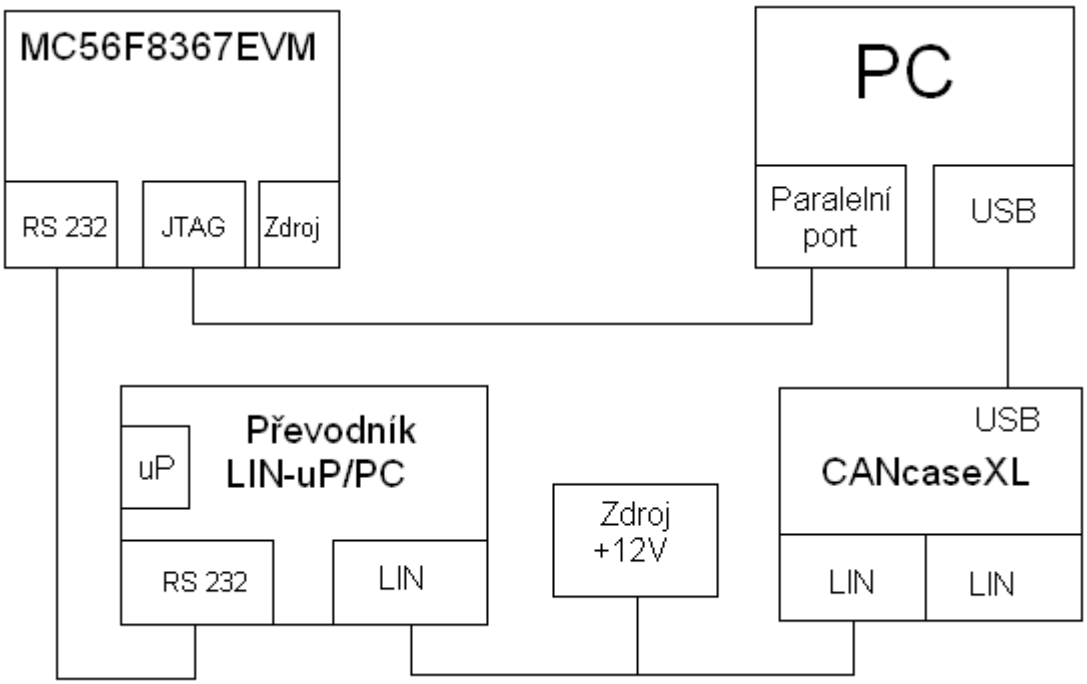
Okno Bean Selektor slouží k vybrání jednotlivých Beanů.Beany lze vybírat v závislosti co nabýzí čip nebo lze vybírat podle kategorií CPU,Interní periferi,externí zařízení nebo softwarové nástroje.

Okno Bean Inspector se zobrazí vždy nastavení jednotlivý beanů.Množství položek pro nastavení,ktetré je možné nastavit se může zvolit pomocí základního,rozšířeného nebo expertního zobrazení.Jak již bylo popsáno v kapitole 3.5, je možné u Beanu nastavit vlastnosti, metody, události.Mnoho vlastností je možné měnit pomocí metod přímo z uživatelského programu.

Posledním zobrazený okno je Errors,v kterém se vypisují chyby v nastavení jednotlivých Beanů.

Pokud je provedeno nastavení,které se může kdykoliv změnit je možné pokračovat v práci na vlastních uživatelských modulech.

5.2 HW konfigurace



Obrázek 17: Hardwarové zapojení

Podrobná zapojení a nastavení

MC56F8367EVM

Zdroj-konektor P3 jack 2.1mm pro napájecí zdroj 12V DC/AC

JTAG-klasický paralelní port 25 pin D-sub

RS232-na desce je vyveden samec DB-9,pro správnou funkci RS 232 je potřeba propojit následující propojky.Konektor JG9 propojit piny 1-2 (TXD0-TXD) a 3-4 (RXD0-RXD).Aktivace celého portu RS232 se provede rozpojením propojky na konektoru JG10.Propojení mezi MC56F8367EVM a převodníkem LIN-uP/PC specifikuje následující Tabulka 3 .

Pin	MC56F8367 EVM	LIN-uP/PC
1	nezapojen	nezapojen
2	TXD	RXD
3	RXD	TXD
4	nezapojen	nezapojen
5	GND	GND
6	nezapojen	nezapojen
7	nezapojen	nezapojen
8	nezapojen	nezapojen
9	nezapojen	nezapojen

Tabulka 3:Propojení mezi MC56F8367EVM a převodníkem LIN-uP/PC

Převodník LIN-uP/PC

Jádro celého převodníku je založeno na obvodu MCP201,což je převodník úrovní z TTL na LIN(Ubat).Obvod je napájen napětí Ubat(6-18V),Ureg slouží jako referenční napětí 5V např. pro mikroprocesor,CS/WAKE=slouží pro přepnutí do nízkopříkonového módu,což je řešeno přepínačem SW1,FAULT/SLPS-volitelný mód přechodu z nízkopříkonového módu s definicí

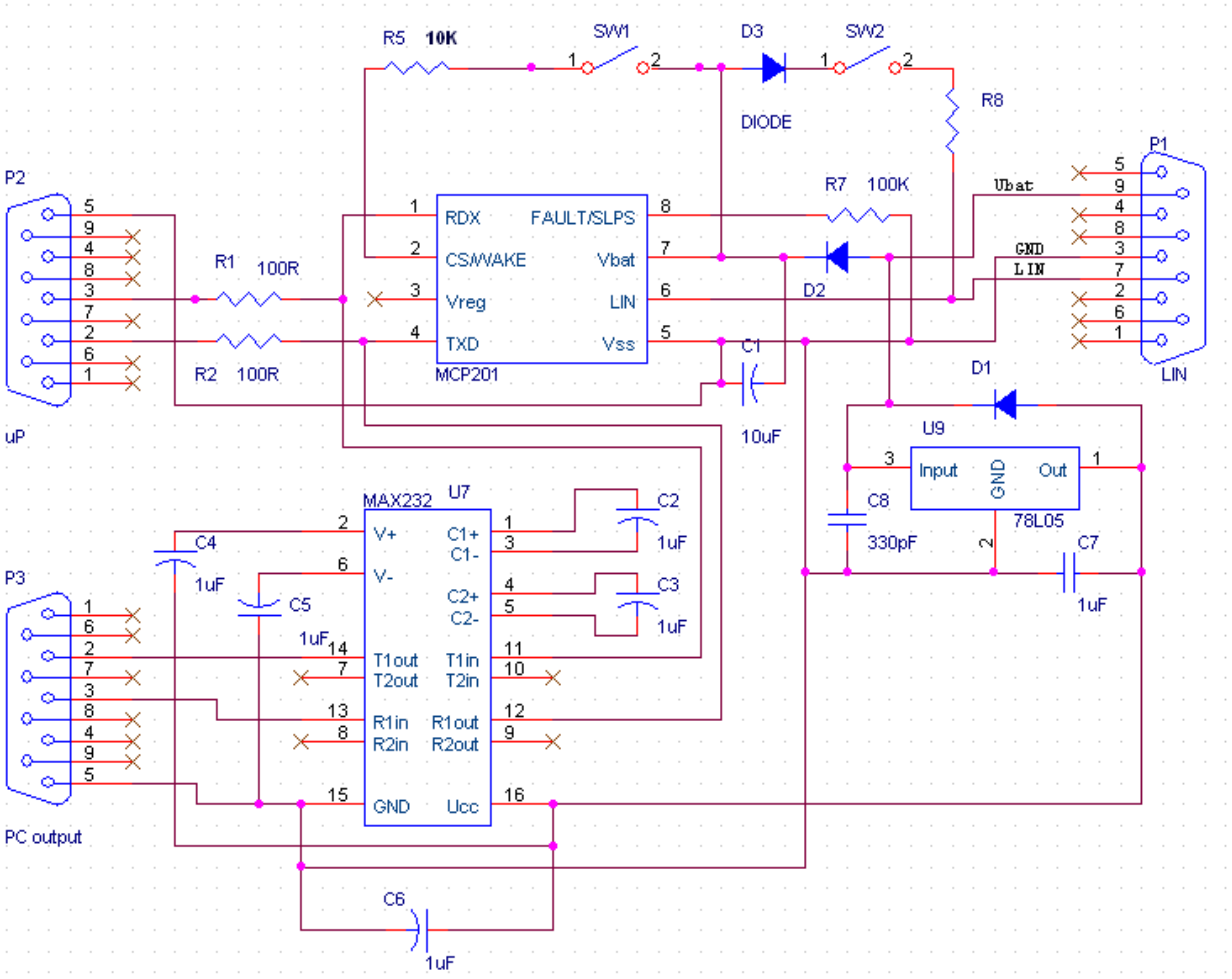
výstupního stavo LIN sběrnice.Přepínač SW2 slouží k nastavení převodníku jako Master (přepínač sepnut) nebo Slave(přepínač rozepnut).

Převodník pracuje obousměrně.Jednak přímý výstup z uP úrovně TTL (P2) ⇔ LIN (P1) ,tak RS232(P3) ⇔ LIN (P1).Pro převod úrovní z RS232 na TTL je použit obvod MAX232,který je napájen stabilizovaným napětím +5V.Schéma je vidět na obrázku 17 převzatého z [7].

Propojení mezi převodníkem LIN-uP/PC a analyzátorem CANcaseXL specifikuje Tabulka 4 .

Pin	LIN-uP/PC	Analyzátor CANcaseXL
1	nezapojen	nezapojen
2	nezapojen	nezapojen
3	GND	GND
4	nezapojen	nezapojen
5	nezapojen	nezapojen
6	nezapojen	nezapojen
7	LIN	LIN
8	nezapojen	nezapojen
9	Ubat	Ubat

Tabulka 4:Propojení mezi převodníkem LIN-uP/PC a analyzátorem CANcaseXL



Obrázek 18: Schéma převodníku LIN-uP/PC

5.3 Popis struktury kódu

Program jsem navrhoval dle specifikace consortia LIN-subbus se sídlem ve Stuttgartu.Na požádání mi zaslali kompletní specifikaci LIN protokolu verze 1.3 [1].

Úloha jednotky Master

- řídí komunikaci na sběrnici
- komunikace se Slave jednotkami:
    - vysílání Synch Break
    - vysílání Synch Field
    - vysílání ID
    - vysílání nebo příjem Data Bytes a Checksum
- řízení módu sítě:
    - vyslání rámce sleep módu z Mastru všem jednotkám Slave
    - příjem nebo vyslání Wake-Up z uzlu Master nebo Slave a zahájení vysílání hlavičky
- obecně:
    - slouží jako reference pro řízení komunikační rychlosti

Kontrola chyb v jednotce Master

Jelikož detekované chyby nejsou signalizovány protokolem LIN, proto jejich zpracování musí být navrženo v daném systému. Zde uvádím nejdůležitější chyby, které je potřeba kontrolovat a následně ošetřit.

Master vysílá:

- kontrola délky rámce Hlavičky a celého rámce zprávy
- bitová chyba

- fyzická chyba na sběrnici
- chyba v zabezpečení Identifikátoru

Master přijímá:

- jednotka Slave neodpovídá
- chyba v zabezpečení dat checksum

Popis struktury kódu

Z předchozího popisu funkce jednotky master je zřejmé, že celou komunikaci na sběrnici zajišťuje jednotka master. Jednotka slave pouze čeká na oslovení patřičným ID pro odeslání dat k masteru, přijetí dat od masteru nebo vyslání WakeUp signálu na sběrnici.

Kód tvořící Master se skládá z:

hlavních funkcí obsažených v souboru Lin.c

- **inicializace** _LINInitialize(), funkce se volá pomocí makra mLINInitialize() .
- **odeslání zprávy** _LINSendMessage(byte LIN_idr, char LIN_size, unsigned int _LIN_fmin, unsigned int _LIN_fmax), funkce se volá pomocí makra mLINSendMessage(tag,i,s), kde tag - je název zprávy, i – ID jednotky, s – počet byte zprávy .
- **příjem zprávy** _LINReceiveMessage(byte tag,byte id,char size), funkce se volá pomocí makra mLINReceiveMessage(tag,i,s), kde kde tag - je název zprávy, i – ID jednotky, s – počet byte zprávy .
- **řízení běhu** LINHandler(void), která je volaná vždy v přerušení časovače TI1, jehož interval přerušení za 1 bit se nastavuje proměnou LIN_INTERRUPT_PERIOD v hlavičkovém souboru Lin_Master.h .

pomocných funkcí obsažených v souboru Lin.c

- **reset a nastavení chybového kódu** _LINResetProtocol(byte code), kde vstupem je hodnota chybového kódu
- **výpočet zabezpečení identifikátoru zprávy** _LINCalcIDParity(LIN_ID LIN_idtr), kde vstupem je odkaz na union LIN_ID, který je v souboru Lin_Master.h .
- **získání ukazatele na začátek bufferu dané zprávy** *_LINGetPointer(unsigned char _LIN_tag, byte _LIN_position), která se volá pomocí makra mLINGetTXPointer(tag) nebo mLINGetRXPointer(tag), kde tag-je názez konkrétní zprávy

Dále v souboru Lin_Master.h jsou definovány prototypy funkcí, uživatelské typy, datové struktury a uniony, definice maker a důležité konstanty.

Důležité konstanty

- LIN_BAUD – definice přenosové rychlosti
- LIN_SLEEP_TIMEOUT – délka prodlevy než na sběrnici nastane sleep mód
- LIN_INTERRUPT_PERIOD - počet přerušovacích period za dobu 1 bitu
- LIN_WAKEUP_BYTE – definice hodnoty WakeUp bytu
- LIN_SYNC_VALUE - definice synchronizační hodnoty

Definice chybových kódů

- LIN_NO_ERROR bez chyb
- LIN_THMIN_ERROR - HEADER nedosáhnutá minimální hodnota
- LIN_THMAX_ERROR - HEADER překročena maximální hodnota
- LIN_TFMIN_ERROR - FRAME nedosáhnutá minimální hodnota
- LIN_TFMAX_ERROR - FRAME překročena maximální hodnota
- LIN_CHECKSUM_ERROR - Chyba v kontrolním součtu
- LIN_DATA_ERROR - Chyba v datech
- LIN_FRAMING_ERROR - Chybný rámec dat

Definice unionů a struktur

- **union LIN_ID** obsahuje strukturu Idbits, která definuje ID jednotky
- **union LIN_MESSAGE_SIZE** obsahuje strukturu SIZEbits, která definice velikost zprávy
- **union LIN_CRC** obsahuje strukturu CRCbits, která definuje hodnotu kontrolního součtu
- **union LIN_STATUS** obsahuje strukturu LINSTS, která obsahuje informační hodnotu o konkrétním stavu proměný TX, RX, ERROR, IDLE, ERROR_BIT0 - ERROR_BIT3
- **union LIN_STATUS1** obsahuje strukturu LINSTS, která obsahuje informační hodnotu o konkrétním stavu proměný WAKEUP, HEADER, FRAME, SLEEP_SENT, WAKEUP_RECEIVED, WAKEUP_SENT, SLEEP_TIMEOUT

V další části souboru Lin_Master.h jsou deklarovány prototypy funkcí a makra (uvedené v kapitole 5.4 Popis API kódu).

5.4 Popis API kódu

Následující makra jsou definována v souboru Lin_Master.h. kde je úplný popis.

Název	Užití	Popis
mLINInitialize	inicializace	inicializace
mLINTXBufferAvailable	vysílání	kontroluje zda je připraven buffer pro vysílání
mLINGetTXPointer	vysílání	nastaví název a vrátí data buffer
mLINSendMessage	vysílání	vysílá zprávu
mLINTXStatus	vysílání	vrací status přenosu
mLINMessageSent	vysílání	kontroluje zda je zpráva vyslána
mLINRXMessageAvailable	příjem	kontroluje zda je připraven přijímací buffer
mLINReceiveMessage	příjem	proces přijímání zprávy od slave
mLINMessageReceived	příjem	kontroluje zda je zpráva přijata
mLINGetMessageTag	příjem	vrací název přijaté zprávy
mLINGetRXPointer	příjem	vrací pointer na přijatá data
mLINRXMessageAvailable	příjem	kontroluje zda je přijatá zpráva dostupná
mLINRXStatus	příjem	vrací status přenosu
mLINCheckWakeUPReceived	ovládání sběrnice	definuje zda byl přijat WakeUp signál
mLINSendWakeUPSignal	ovládání sběrnice	definuje zda byl odeslán WakeUp signál
mLINSleepTimeOut	ovládání sběrnice	detekování režimu Sleep

Tabulka 5: Použití maker

mLINInitialize()

Podminka: ---

Vstup: ---

Vystup: 0 = inicializace v pořádku , 0!= chyba v inicializaci

Souhrn: ---

mLINTXBufferAvailable()

Podminka: mLINInitialize()

Vstup: ---

Vystup: 1 = buffer připraven pro vyslání dat , 0 = buffer nepřipraven pro vyslání dat

Souhrn: ---

mLINGetTXPointer(tag)

Podminka: mLINInitialize() , mLINTXBufferAvailable()

Vstup: tag -název zprávy

Vystup: vrací data pointer na TX buffer

Souhrn: Podle názvu zprávy vrátí data buffer, kde je uložena hodnota pro vysílání

mLINSendMessage(tag,i,s)

Podminka: mLINInitialize()

Vstup: tag - název zprávy , i - ID jednotky , s - velikost zprávy v bytech

Vystup: ---

Souhrn: Pro poslání zprávy

mLINTXStatus(tag)

Podminka: Po skončení vysílání

Vstup: tag -název zprávy

Vystup: vrací chybový kód

Souhrn: ---

mLINMessageSent(tag)

Podminka: mLINSendMessage

Vstup: tag - název zprávy

Vystup: 1 = přenos dokončen , 0 = přenos nedokončen

Souhrn: ---

mLINRXBufferAvailable()

Podminka: mLINInitialize()

Vstup: --

Vystup: 1 = buffer připraven pro příjem dat , 0 = buffer nepřipraven pro příjem dat

Souhrn: ---

mLINReceiveMessage(tag,i,s)

Podminka: mLINInitialize()
mLINRXBufferAvailable()

Vstup: tag -název zprávy , i = ID zprávy , s = velikost zprávy

Vystup: ---

Souhrn: Vyšle žádost o data od slave jednotky

mLINMessageReceived(tag)

Podminka: mLINReceiveMessage(tag,i,s)

Vstup: tag - název zprávy

Vystup: 1 = zpráva byla přijata , 0 = zpráva ještě nebyla přijata

Souhrn: Zjistí podle názvu jestli zpráva byla nebo nebyla přijata

mLINGetMessageTag()

Podminka: mLINInitialize()

Vstup: ---

Vystup: Vrací název přijaté zprávy

Souhrn: ---

mLINGetRXPointer(tag)

Podminka: mLINInitialize() , mLINRXBufferAvailable()

Vstup: tag -název zprávy

Vystup: Vrací data pointer na buffer

Souhrn: Makro vrací data pointer pro čtení zprávy,která je identifikovaná podle názvu zprávy

mLINRXMessageAvailable()

Podminka: mLINReceiveMessage(tag,i,s)

Vstup: ---

Vystup: 1 = zpráva byla vyslána , 0 = zpráva nebyla vyslána

Souhrn: Kontroluje jestli je schopná přijímat data,ale předtím se musí zkontrolovat jestli nenastaly chyby

mLINRXStatus(tag)

Podminka: Po skončení příjmu dat

Vstup: tag -název zprávy

Vystup: vrací chybový kód

Souhrn:

mLINCheckWakeUPReceived()

Podminka: ---

Vstup: ---

Vystup: 1 = wake-up signál byl přijat , 0 = wake-up signál nebyl přijat

Souhrn: Kontrola jestli byl nebo nebyl přijat WakeUp signál

mLINSendWakeUPSignal()

Podminka: LINInitialize().

Vstup: ---

Vystup: ---

Souhrn: Vyšle WakeUp signál na sběrnici

mLINSleepTimeOut()

Podminka: mLINMIntInitialize()

Vstup: 1 = čas překročen , 0 = čas nepřekročen

Souhrn: Kontroluje překročení času,po které vyšle master všem slave jednotkám příkaz, aby přešly do Sleep módu

5.5 Ukázkový příklad použití

Pošle zprávu k slave

```
if(mLINInitialize())
{
  LED1_Toggle();                                  //pokud je chyba ledka změní stav
}
else
{
        if(mLINTXBufferAvailable())               // kontroluje zda je dostupny TX buffer
        {
                st=mLINGetTXPointer(1);           // získá pointer pro zprávu
                *st=Data_send[0];                 //vložení dat
                st++;
                *st=Data_send[1];                 //vložení dat
                st++;
                *st=Data_send[2];                 //vložení dat
                st++;
                *st=Data_send[3];                 //vložení dat
        }

        mLINSendMessage(1,0x05,4);                // vyšle zprávu 1,k slave s ID=0x05,4Byte

        while(mLINMessageSent(1)==0)              // čekání dokud se nevyšle kompletní zpráva
        ;
        if((ErrorCode=mLINTXStatus(1))!=0)        // kontrola detekce erroru
        {
         LED1_Toggle();                           //pokud je chyba ledka změní stav
        }

}
```

Přijme zprávu od slave

```
if(mLINInitialize())
{
  LED1_Toggle();                                  //pokud je chyba ledka změní stav
}
else
{
        if (mLINRXBufferAvailable())              // kontroluje zda je RX buffer připraven
        mLINReceiveMessage(2,0x01,4);             // žádá o data zpráva2,ID=0x01 a velikost=4

        while(mLINMessageReceived(2)==0)          //čeká dokud zpráva není přijata
        ;
        if((ErrorCode=mLINRXStatus(2))!=0)        // kontrola detekce chyby v příjmu
        {
          LED1_Toggle();                          //pokud je chyba ledka změní stav
        }
                Tag=mLINGetMessageTag();          // získá název přijaté zprávy
                pt=mLINGetRXPointer(Tag);         // získá pointer na data
                *pt++;
                *pt++;
                Data_send[0]=*pt;                 // uloží přijatou zprávu do
                pt++;                             // pole Data_send
                Data_send[1]=*pt;
                pt++;
                Data_send[2]=*pt;
                pt++;
                Data_send[3]=*pt;
        }
}
```

Vyšle sleep rámec pokud je překročen čas 25000 bit-time

```
if(mLINSleepTimeOut())                            // pokud je překročena hranice pro sleep mod
{
 if(_LINStatus1.LINSTS.SLEEP_SENT==0)
 {
        if(mLINTXBufferAvailable())               // kontroluje zda je dostupny TX buffer
        {
                st=mLINGetTXPointer(3);           // získá pointer na zprávu
                *st=0x00;                         //vložení dat
```

```
            st++;                                   // první data Byte musi být 0x00 pro sleep
            *st=0x00;                               //vložení dat

        }
        mLINSendMessage(3,0x3C ,2);                 // vyšle sleep všem jednotkám slave
        while(mLINMessageSent(3)==0)                // čeká na vyslání
        ;

        if((ErrorCode=mLINTXStatus(3))!=0)          // kontrola pokud byl detekován error
        {
        LED1_Toggle();                              //pokud je chyba ledka změní stav
        }
        _LINStatus1.LINSTS.SLEEP_SENT=1;
  }
}
```

Kontroluje pokud byl Wake-Up poslán od slave

```
if(mLINCheckWakeUPReceived())                       // kontroluje zda byl přijat WakeUp
{
  if(_LINStatus1.LINSTS.SLEEP_SENT==1)              // pokud byl vyslán sleep rámec
  {
        mLINSendWakeUPSignal();                     // vyšle WakeUp signál
        CompleteTX();                               // čeká na dokončení vyslání
        _LINStatus1.LINSTS.WAKEUP_SENT=0;// nulování proměné WakeUP_SENT
        _LINStatus1.LINSTS.SLEEP_SENT=0;            // nulování proměné SLEEP_SENT
  }
  else
  {
    LED1_Toggle();                                  //pokud je chyba ledka změní stav
  }
}
```

5.6 Testování jednotky Master

První testování jednotky Master probíhalo bez převodníku úrovní na LIN sběrnici.Jelikož vývojová deska MC56F8367EVM má výstupní rozhraní RS-232, bylo možné první testy provádět pomocí sériové komunikace s počítačem.Aby se mohly přijímat a vysílat znaky musel jsem nainstalovat na počítač program EasyCom, který sleduje sériovou linku RS-232 a dokáže přijímat a vysílat znaky.

Dalším krokem testování jednotky Master bylo pomocí analyzátoru CANcaseXL.Po prostudování nanuálu o CANcaseXL a jeho následném nastavení analyzátor nepřijímal žádné data, proto jsem nejdříve začal kontrolovat formát zprávy softwarově pomocí časovače s nastavováním příznakových bitů.V této fázi jsem naprogramoval diagnostiku pro chod programu, což je možné využít v reálné aplikaci, kde jednotka pošle nebo se z jednotky vyčte chybový kód. Pro konečnou kontrolu jsem celou zprávu a vysílání ostatních stavů jako je Sleep rámec a WakeUp signál kontroloval pomocí osciloskopu podle specifikace LIN 1.3 .

Po ověření správnosti formátu rámce zprávy na osciloskopu už jen zbývalo to samé otestovat pomocí LIN anylyzátoru CANcaseXL s připojeným převodníkem. Dlouhou dobu jsem se pokoušel rozběhnout analyzátor, který mi nechtěl číst data od Masteru.Nakonec mi poslal p.Švimberský, který s LIN analyzátorem pracoval, nastavení a poradil mi zkusit z instalačního CD nahrát jiné drivery pro CANcaseXL.Po kompletním přeinstalování začal LIN analyzátor pracovat , ale nečekaně začal přijímat chybné rámce.Po shlédnutí průběhu osciloskopem jsem zjistil, že přestal fungovat převodník úrovní z TTL(5V) na LIN(Ubat).V této fázi, kdy je jednotka LIN Master otestována pomocí osciloskopu další testování pomocí LIN analyzátoru skončilo,protože převodník z TTL na LIN úrovně není k sehnání a termím dodání se pohybuje kolem 3-5 týdnů.

Kapitola 6

Závěr

Tato práce ukazuje nové možnosti programování a vývoje embedded aplikací hlavně pro automobilový průmysl.V dnešní době s rozvojem elektrických zařízeních v automobilech se stále zvyšujícími nároky na rychlost, spolehlivost a rozměry těchto zařízení, je nutný vývoj nových metod pro tvorbu těchto zařízení.

Jeden z mnoha možných nástrojů je Processor Expert učený pro procesory Freescale, jehož použitím lze radikálně zrychlit vývoj vestavěných aplikací. Processor Expert definuje tzv. Embedded Beans, které obsahují základní sadu informací o procesoru. Díky tomu se snižuje čas realizace celého zařízení, protože programátor může začít pracovat na tvorbě programu, aniž by měl fyzicky k dispozici konkrétní procesor se svými specifickými odlištnostmi.

Tyto nástroje jsou využity při implementaci kódu jednotky Master pro sběrnici LIN na vývojové desce MC56F8367EVM od Freescale.Tato vývojová deska díky hotovým příkladům, umožňuje rychlé nasazení do konkrétní aplikace.

Sběrnice LIN popsaná v kapitole 2 byla v této práci popsána podle protokolu LIN 1.3 .V dnešní době se je již protokol verze LIN 2.0, který definuje nové vlastnosti a časy jednotlivých délek stavů. Neocenitelnou výhodou je implementace na jakýkoliv procesor s obvodem UART, tím se realizace velice zlevňuje.

Nejdůležitější částí celé práce byla implementace a otestování sotwaru tvořící jednotku Master pro LIN. Potom jsem popsal jeho API a pro ukázku jsem vytvořil demonstrační příklady.

Kapitola 7

Použitá literatura

[1] LIN Consortium, LIN Specification 1.3, http://www.linsubbus.org

[2] Processor Expert [online] , http://www.processorexpert.com.

[3] Freescale Semiconductor [online] , http://www.freescale.com.

[4] Microchip Technology Inc [online] , www.microchip.com

[5] Burkhard Mann , C pro mikrokontroléry,BEN-technická literatura 2004

[6] Antonín Vojáček , LIN - Local Interconnect Network , http://www.hw.cz

[7] Švimberský Zdeněk , Bakalářská práce , 2006

[8] Vector CANtech, Inc.[online] , www.vector-worldwide.com

Příloha

Obsah přiloženého CD

Přiložené CD v jednotlivých složkách obsahuje:

Bakalářská práce – Tuto bakalářskou práci ve formátu PDF

Dokumentace – Dokumentace k používanému hardwaru a softwaru

Obrázky – Veškeré použité obrázky

Projekt – Kompletní projekt jednotky master pro LIN včetně zdrojových souborů Lin.c a Lin_Master.h