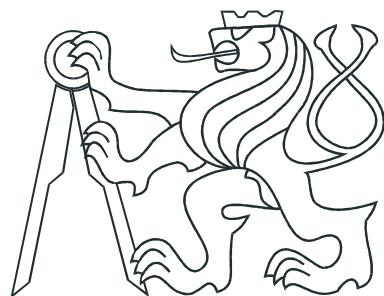


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ



DIPLOMOVÁ PRÁCE

**Distribuovaný varovný systém pro
automobilovou bezpečnost**

Praha, 2007

Autor: Jan Soukup

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jan Soukup

Obor: Technická kybernetika

Název tématu: Automobilový distribuovaný varovný systém

Zásady pro výpracování:

1. Seznamte se s mikroprocesorovou a nízkofrekvenční vysílací technikou.
2. Navrhněte komunikační protokol pro distribuovaný varovný systém.
3. Naprogramujte aplikaci pro tento varovný systém.
4. Otestujte a zdokumentujte.

Seznam odborné literatury:

Xue Yang: A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning. University of Illinois at Urbana-Champaign. - (http://research.microsoft.com/~zhao/pubs/yang_x_v2v.pdf)

Vedoucí diplomové práce: Ing. Jan Krákora

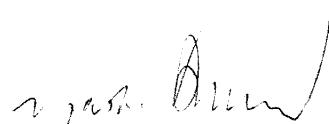
Termín zadání diplomové práce: zimní semestr 2005/2006

Termín odevzdání diplomové práce: leden 2007

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry



prof. Ing. Vladimír Kučera, DrSc.
děkan



Poděkování

Chtěl bych poděkovat zejména vedoucímu práce ing. Janu Krákorovi za trpělivé vedení při práci a podporu, kterou mi při naší spolupráci poskytoval.

Dále děkuji všem, kteří mi byli oporou v mém studiu na ČVUT a především rodičům, kteří mi toto studium umožnili.

Abstrakt

Tento projekt se věnuje tématu preventivního varování řidičů v nebezpečných situacích na silnici a řadí se tak mezi prvky aktivní bezpečnosti v automobilech. V první fázi jsme se zabývali průzkumem na poli automobilové bezpečnosti. V závislosti na něm byl pak rozpracován vlastní návrh, který si klade za cíl, narozdíl od ostatních projektů, minimalizovat pořizovací náklady a v budoucnu se tak prosadit na trhu jako systém pro levnější a starší vozy a být schopný spolupracovat s ostatními systémy. Vybrali jsme a navrhli hardwarové prostředky a připravili jsme komunikační protokol pro tento varovný systém. Výsledky práce jsme vyzkoušeli reálně na zhotovené aplikaci.

Abstract

This work is involving into a driver's preventive warning system in dangerous traffic situations and classifies itself as a car's active safety element. At first we have made a research on the automotive safety field. According to this research we have started our own project which prescribes his main goal, in contrast to other projects, to minimize the production costs and to achieve the market as a low-cost system in the future. Therefor we have to fulfill the demand on the cooperation with other projects. We have chosen and made a hardware for this system and we have prepared a communication protocol for the wireless communication. The results of this work were tested with a real application.

Obsah

Seznam obrázků	ix
Seznam tabulek	xi
1 Úvod	1
1.1 Návrh řešení	3
2 Komunikační protokol	7
2.1 Rozbor problému	7
2.2 Propustnost bezdrátové sítě	9
2.3 Požadavky na komunikační protokol	10
2.4 Návrh protokolu	11
2.5 Vysílací část protokolu	12
2.5.1 Princip návrhu	12
2.5.2 Podrobný popis funkce	13
2.5.2.1 <i>Překážka</i>	15
2.5.2.2 <i>Auto</i>	16
2.5.2.3 <i>Vysílač1</i>	17
2.5.2.4 <i>Vysílač2</i>	18
2.5.2.5 <i>Transmission</i>	18
2.5.3 Verifikace	19
2.6 Přijímací část protokolu	20
2.6.1 Princip návrhu	20
2.6.2 Podrobný popis funkce	21
2.6.2.1 <i>PowerOn</i>	21
2.6.2.2 <i>ReceiveCheck</i>	22
2.6.2.3 <i>Message</i>	22

2.6.3	Verifikace	23
2.7	Zhodnocení	24
3	Hardware	25
3.1	GPS + ACC Board	25
3.1.1	Základní popis	25
3.1.2	Napájení	27
3.1.3	Popis desky plošného spoje	28
4	Software	31
4.1	TinyOS	31
4.1.1	Úvod	31
4.1.2	Komponenty	34
4.1.3	Konfigurace TinyOS	37
4.2	Vlastní návrh	39
4.2.1	Aplikace pro Tmote Sky	39
4.2.1.1	Struktura programu	40
4.2.1.2	Stručný úvod do GPS	42
4.2.1.3	Komunikace s GPS modulem	45
4.2.1.4	Komunikace s akcelerometrem	48
4.2.1.5	Bezdrátová komunikace mezi moduly Tmote Sky	51
4.2.2	Aplikace pro PC	54
5	Rozšiřující Hardware	58
5.1	Řídící jednotka	58
5.1.1	Napájení	60
5.1.2	Procesor	62
5.1.3	Radiový modul	64
5.1.4	Ethernet	65
5.1.5	CAN	66
5.1.6	USB rozhraní	67
5.1.7	Seriová rozhraní k PC	68
5.1.8	Rozšiřující konektory, uživatelské vstupy/výstupy	69
6	Závěr	72

Literatura	75
A Elektronické schéma GPS/ACC modulu	I
B Elektronické schema rozšiřující Řídící Jednotky	III
C Praktické připomínky k prostředí Eclipse	VII
D Obsah přiloženého CD	IX

Seznam obrázků

1.1	Systém distribuce informací v praxi	3
1.2	Tmote Sky	5
2.1	Rozbor situace	8
2.2	Rozbor složitější situace	8
2.3	Graf závislosti zpoždění zpráv na počtu jednotek v síti	10
2.4	Příklad stavového automatu	11
2.5	Blokové schéma vysílače	13
2.6	schéma pro detekci nebezpečného chování	15
2.7	Model Auto	16
2.8	Model Vysilac1	17
2.9	Model Vysilac2	18
2.10	Model Transmission	18
2.11	Blokové schéma přijímače	21
2.12	Model PowerOn	21
2.13	Model ReceiveCheck	22
2.14	Model Message	23
3.1	Blokové schema navrženého modulu	25
3.2	Napájení navržené desky	27
3.3	Schéma připojení GPS modulu	28
3.4	Schéma zapojení akcelerometru	29
3.5	Navržený plošný spoj desky	30
3.6	Vyrobená deska GPS/ACC	30
4.1	struktura programování pod TinyOS	32
4.2	Bezdrátové moduly pro TinyOS	33
4.3	Provázání komponent v konfiguračním souboru	34

4.4	Přenos dat do PC	39
4.5	Blokové schéma modulu Tmote Sky	40
4.6	Zpráva NMEA	43
4.7	Výstupní zprávy protokolu NMEA	44
4.8	Vstupní zprávy protokolu NMEA	44
4.9	Softwarový nástroj SiRFDemo	45
4.10	Arbitráž sdílených hardwarových prostředků	46
4.11	Výstupní PWM signál akcelerometru	50
4.12	Formát zpráv TOS_Msg	52
4.13	Aplikace SerialForwarder	54
4.14	Grafické okno aplikace pro PC	55
4.15	Inicializace SF protokolu	56
5.1	Blokové schéma zapojení	59
5.2	Napájecí obvod	61
5.3	Zapojení procesoru	62
5.4	Připojení modulu ZigBee	64
5.5	Připojení ethernetového rozhraní	66
5.6	Připojení sběrnice CAN	66
5.7	Připojení USB rozhraní	68
5.8	Zapojení seriových rozhraní	69
5.9	Indikační LED diody	70
5.10	Tlačítka	70
5.11	Zapojení převodníku 5V/3,3V	71
A.1	Schéma navrhnutého modulu GPS/ACC	II
B.1	Schéma mikroprocesorové části	IV
B.2	Schéma rádiového modulu	V
B.3	Schéma ethernetového budiče	VI
C.1	TinyOS plugin pro Eclipse	VII
C.2	Graf aplikace v prostředí Eclipse	VIII

Seznam tabulek

1.1 Srovnání bezdrátových technologií	4
---	---

Kapitola 1

Úvod

V současné době je v silničním provozu značným problémem velké množství dopravních nehod. Ačkoliv již existuje velké množství prvků aktivní bezpečnosti dávající řidiči podporu při řízení, jmenujme například systémy ABS (Anti Block System), ESP (Electronic Stability Program), stále na trhu chybí systém, který by s předstihem informoval řidiče o silniční situaci a okolních jevech. Jeho úkolem je zabránit dopravní nehodě a v případě, že to není možné, pomoci zmírnit následky této nehody. Požadavky na tento systém můžeme shrnout do následujících bodů:

- informovat řidiče co nejdříve o dopravní situaci,
- v případě nebezpečí varovat a
- aktivně asistovat nebo zasahovat do řízení tak, aby nedošlo k dopravní nehodě.

Zároveň ale tento systém pomáhá řidiči

- udržovat bezpečnou rychlosť,
- udržovat bezpečný odstup,
- sledovat jízdní pruh,
- bezpečně projízdět křižovatky,
- v neposlední řadě zmírnit následky nehody pokud k ní dojde.

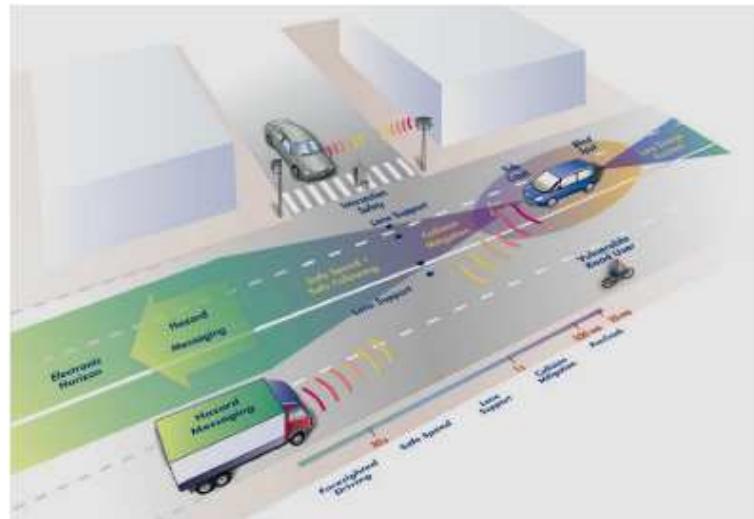
V současné době je ve vývoji několik takovýchto systémů, jedním z nejvýznamnějších je společný projekt předních evropských společností v automotive průmyslu a Evropské Unie zvaný *PReVENT*. Tento projekt by měl být oficiálně představen na podzim roku

2007. Je složen z mnoha podprojektů používajících společnou hardwarovou platformu, ale zabývajících se specifickým problémem. To jest například sbírání informací z okolních vozů a vozovky, laterální podpora řidiče při řízení (ve smyku), zmírnění následků nehody atd.

Již delší dobu se pracuje na standardizaci komunikace pro bezdrátovou síť v automobilovém průmyslu. Tyto standardy ale stále nejsou dokončeny a momentálně jsou na trhu dva. Prvním z nich je projekt *DSRC* (*Dedicated Short Range Communication*) vznikající v Evropě a druhým pak *WAVE* (*Wireless Access in Vehicular Communication*) vyvíjený americkými společnostmi. Oba projekty nejsou momentálně kompatibilní. Především se liší v bezdrátové technologii, kdy oba systémy používají jiné frekvenční pásmo. Kvůli vzájemné provázanosti obou trhů bude ale nutné obě technologie přizpůsobit, na čemž se dle dostupných pramenů (Schulze, n.d.) pracuje.

Vyvíjené systémy jsou velice komplexní a integrují v sobě mnoho technologií a tím pádem bude jejich cena pro řidiče starších vozů zatím velmi nepříznivá. Proto vznikl na katedře Řídící techniky ČVUT FEL projekt, který si dává za úkol zhodnotit zařízení schopná podobné komunikace mezi vozidly a varovat řidiče nebo i podpůrné systémy ve vozidle před případnou nebezpečnou situací. Zároveň ale myšlenka celého projektu spočívá navíc v tom, že výrobek by byl cenově mnohem dostupnější než výše zmíněné produkty. Navržená bezdrátová technologie odpovídá evropskému standardu *DSRC* a komunikuje na frekvenci 2,4GHz. K určování aktuální polohy slouží GPS modul a pro základní detekci překážek je použito akcelerometru, který zaznamenává náhlé změny směru nebo prudké brždění.

V budoucnu by mohl náš navržený systém existovat paralelně a spolupracovat se systémem *PReVENT* tak, aby se vzájemně doplňovaly. Vzhledem k předpokládané ceně obou systémů by mohl každý oslovit jinou cílovou skupinu. Zatím bohužel podrobnosti o technickém pozadí projektu *PReVENT* zůstávají chráněny, a tak nelze z dostupných informací navrhnut plně kompatibilní systém. Další podrobnosti o projektu *PReVENT* lze nalézt na internetovém zdroji viz (EU, 2007). Na obrázku 1.1 je naznačen příklad komunikace mezi vozidly v praxi.



Obrázek 1.1: Systém distribuce informací v praxi

1.1 Návrh řešení

Na začátku byl plán této diplomové práce navrhnout výše popsaný systém preventivní bezpečnosti. To se ale v průběhu vzhledem k rozsáhlosti celého projektu ukázalo být příliš náročné. Zároveň se s přibývajícími informacemi o dané tematice vyvíjely částečně i požadavky kladené na tento systém. Ty můžeme shrnout do následující několika bodů, systém by měl samostatně pracovat tak, aby byl schopen:

- vyhodnocovat nebezpečné situace,
- co nejrychleji varovat při detekci problému,
- spolupracovat se systémy sběrnice CAN,
- komunikovat s ostatními vozy pomocí bezdrátové technologie ZigBee,
- určovat polohu systémem GPS,
- měřit zrychlení ve dvou osách,
- zajistit výrobní náklady na zařízení.

V závislosti na výše uvedených požadavcích jsem navrhl komunikační protokol popsaný v kapitole 2. Je založen na článccích a studiích uvedených v referencích v příslušné kapitole. Sestavil jsem komunikační model pomocí nástroje UPPALL a provedl verifikaci návrhu.

V dalším kroku jsem vybral a navrhl hardwarové prostředky. Nejprve si bylo potřeba rozmyslet, jakým způsobem bude systém určovat polohu vozidel, která je pro tento systém nezbytná. Původní záměr počítal s technologií, která by nahradila systém GPS a určovala přibližnou pozici vozidel. Toho se mělo dosáhnout použitím „informačních patníků“, které by byly rozmístěny podél vozovky a vysíaly by úzký směrový signál s aktuální polohou a časem. Kolem jedoucí vozy by si aktualizovaly tato data a podle vypočtené průměrné rychlosti by zjišťovali přibližnou polohu mezi těmito vysílači. Toto řešení by mělo rozhodně vliv na cenu koncového zařízení, kde nejdražším prvkem je modul GPS. Musela by ale být v rámci projektu vybudována rozsáhlá infrastruktura vysílačů. Po zvážení této skutečnosti, vypracování technického pozadí a po zamýšlení nad celkovou složitostí a přesnosti, jsme se rozhodli od této části ustoupit a použít klasický systém GPS.

Dalším významným bodem v projektu byl výběr technologie pro bezdrátový přenos dat. Bylo k dispozici několik návrhů, které zahrnovaly systémy WiFi, GPRS/GSM, Bluetooth a ZigBee. V tabulce 1.1 je vidět přehled základních vlastností těchto technologií.

Tabulka 1.1: Srovnání bezdrátových technologií

Obchodní název Standard	GPRS/GSM 1xRTT/CDMA	Wi-Fi 802.11b	Bluetooth 802.15.1	ZigBee 802.15.4
Aplikační zaměření	Hlas a Data	Web, Email, Video,	Náhrada za kabel	Monitorování, Řízení
Systémové zdroje (paměť)	16MB a více	1MB a více	250KB a více	4KB - 32KB
Životnost baterií (dny)	1 - 7	0.5 - 5	1 - 7	100 - 1 000
Maximální velikost sítě (počet uzlů/sítě)	1	32	7	65 000
Přenosová rychlosť (Kb/s)	64 - 128	11 000	720	20 - 250
Komunikační dosah (m)	1 000 i více	1 -100	1 - 10	1 - 100
Výhody	Dosažitelnost, Kvalita	Rychlosť, Flexibilita	Cena, Jednoduchost	Spolehlivost, Výkon/Cena

Jako nevhodnějšího kandidáta jsme vybrali ZigBee vzhledem k jeho vlastnostem a ceně. Důležitým údajem je také spotřeba energie, která u této technologie je bezkonkurenčně nejnižší. Základní informace o ZigBee lze získat v tomto článku (Vojáček, 2006) spolu s odkazi na další literaturu.

Jako hardwarovou platformu pro bezdrátový přenos dat jsme si vybrali zařízení Tmote Sky od společnosti Moteiv s chipem ZigBee od firmy Chipcon. Pro tyto, a i další, moduly je určen speciální operační systém TinyOS, se svým programovacím jazykem nesC, který bude popsán v další kapitole. Tmote Sky jsou osazeny mikroprocesorem MSP430 od firmy Texas Instruments s dostatečnou kapacitou paměti Flash (48kB) a RAM (10kB) oproti ostatním zařízením. Lze dohledat v datasheetu (Mote, 2006) nebo podrobnější info v datasheetu mikroprocesoru (TI, 2006). V současné době má katedra k dispozici deset kusů těchto modulů.



Obrázek 1.2: Tmote Sky

K tomuto zařízení jsme vyrobili rozšiřující desku osazenou modulem GPS a akcelerometrem, která je podrobně popsána v kapitole 3, a otestovali jsme komunikaci mezi oběma zařízeními.

Pro tato zařízení jsem vytvořil aplikaci v jazyce nesC. Celý systém přenáší data z GPS modulu a akcelerometru do modulu Tmote Sky, následně bezdrátově do dalšího modulu, který je připojen přes USB port do PC. Na PC je pak vytvořen lokální server, kam se přenáší data z připojeného Tmote Sky modulu a jsou zobrazována v grafické aplikaci. Navržená aplikace spolu s programovacím jazykem nesC a operačním systémem TinyOS je popsána v kapitole 4.

V průběhu realizace jsme dále zjistili, že ačkoliv jsou moduly Tmote Sky vhodné pro náš systém, postrádají podporu sběrnice CAN. Z tohoto důvodu jsme se rozhodli navrhnout novou řídící jednotku. Z tohoto důvodu jsem navrhl testovací desku, kde je implementován modul ZigBee a komunikace CAN. Dále jsou na desce vyvedena rozšiřující rozhraní RS232, Ethernet, USB a další. Tato řídící jednotka vychází z předchozích diplomových prací (Musil, 2003) a (Černý, 2005). Je navrhnuta s novějším procesorem firmy Atmel AT91SAM7XC a zachovává hardwarovou kompatibilitu s hotovými moduly ke starší řídící jednotce. Návrh je popsán v kapitole 3.

Kapitola 2

Komunikační protokol

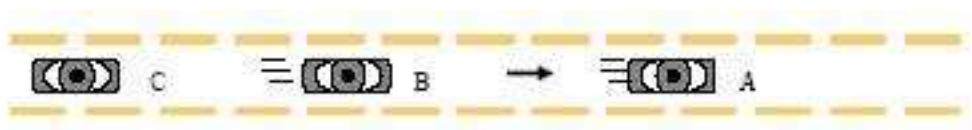
2.1 Rozbor problému

Pro komunikaci mezi vozidly s prediktivním varovným systémem je kritickým problémem časové zpoždění při předávání zpráv. Představme si situaci na dálnici, kdy je výrazně snížena viditelnost na silnici, například mlha nebo sněžení, a že se 100 metrů před vozidlem utvořila kolona. Pokud zkombinujeme tedy nepříznivé podmínky s vysokou rychlostí na dálnicích a často i s nepozorností řidičů může velice lehce dojít k hromadným haváriím s tragickými následky.

Systém by měl tedy varovat dostatečně včas řidiče před blížícím se nebezpečím, ať už při špatné nebo dobré viditelnosti a dále pak varovat řidiče pohybující se za automobilem, pokud se tak již nestalo. Stejně tak je nutno předejít nesprávnému varování, které by mohlo vyvolat další nebezpečnou situaci. To znamená, že je v komunikaci potřeba rozlišit jízdní pruhy a směry vozidel. Toho lze dosáhnout při komunikaci ZigBee použitím adres jednotlivých prvků sítě a správnou implementací kódu.

Dále by měl systém být schopen varovat při náhlých změnách rychlosti nebo směru automobilů pohybujících se v bezprostřední blízkosti. Studie totiž ukazují, že pravděpodobně 60-ti % automobilových nehod by mohlo být zabráněno, kdyby byl řidič upozorněn alespoň půl sekundy před kolizí. Vezměme si případ z obrázku 2.1.

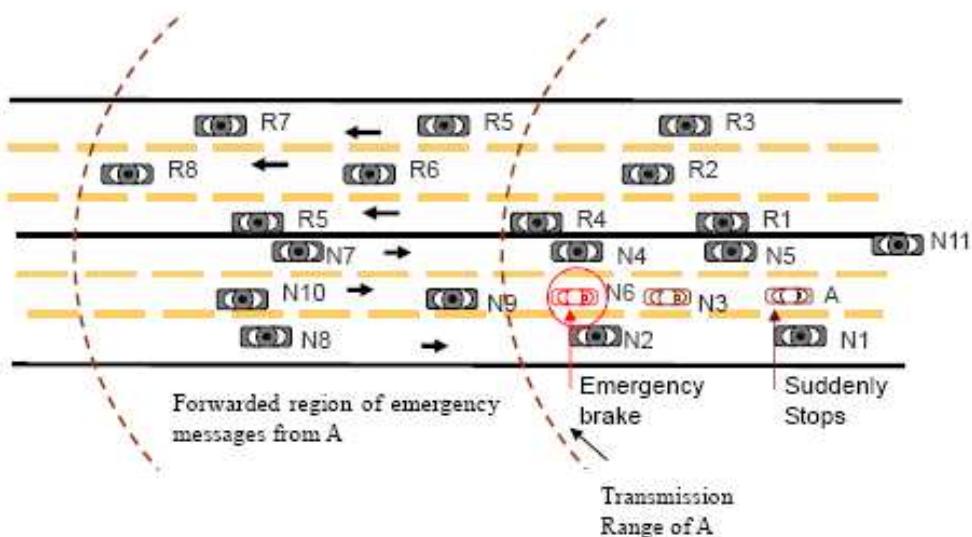
Uvažujme, že všechny auta se pohybují rychlostí 35m/s (126km/hod). Pokud auto A uvidí překážku a náhle zabrzdí, řidič auta B uvidí brzdová světla. V závislosti na lidských reakcích, které se pohybují u člověka v rozmezí 0,7 - 1,5s, řidič vozu B začne brzdit. Řekněme s reakcí 1s. Pokud je tedy vzdálenost obou vozidel menší než 35m může dojít k nehodě. Zapojíme do schématu vůz C a předpokládáme, že řidič toho vozu nevidí



Obrázek 2.1: Rozbor situace

brzdrová světla vozu A. Jeho reakce bude také zpožděna o 1s. Takže začne-li A brzdit, vůz C se 2s pohybuje stále stejnou rychlostí a s největší pravděpodobností, pokud vzdálenost vozu C a B je menší než 35 metrů nebo C a A je menší než 70 metrů, dojde pravděpodobně k nehodě.

Pokud by A mohl bezprostředně začít vysílat varovný signál, který by oba následující vozy zachytily s velmi malým zpožděním, C by měl poměrně velkou šanci, jak předejít nehodě a B by z této situace vytěžil například při špatné viditelnosti nebo při nepozornosti.



Obrázek 2.2: Rozbor složitější situace

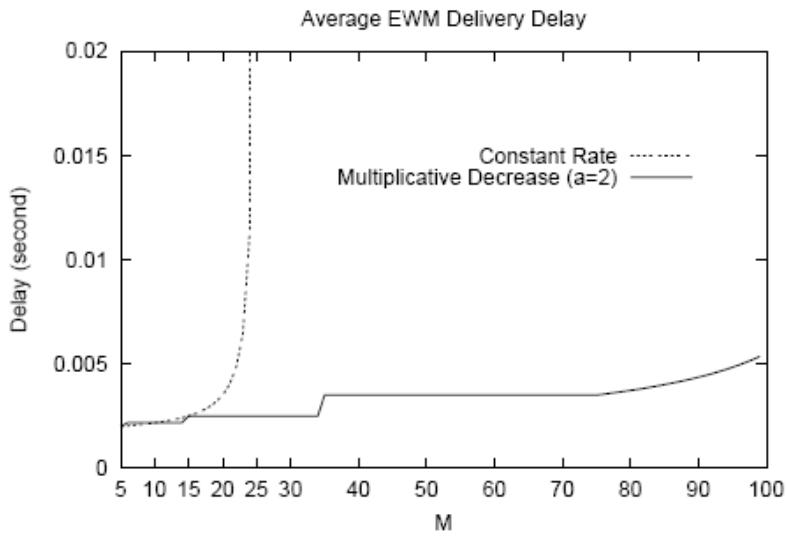
Na obrázku 2.2 je znázorněn komplexněji předchozí problém. Je vidět, že pokud vůz A se stane původcem vysílaného signálu, je potřeba v prvé řadě varovat vozy nacházející se bezprostředně za ním, tedy N3, N6, N9 a N10. Ostatní vozy nemusí být varovány pokud vůz A nevybočí nijak ze svého jízdního pruhu. Vozidla N9 a N10 už ale nejsou ve vysílacím dosahu původce varovného signálu A. Pokud by N6 reagoval na dané varování příliš prudce, dostali bychom se opět k problému z obrázku 2.12. Proto může vůz N3 nebo

N6 zachycený signál dále předávat a rozšířit tak pole dosahu varování. Tím ale vznikne v síti redundancy a je třeba vymyslet mechanismus, který zajistí dobrou propustnost sítě.

2.2 Propustnost bezdrátové sítě

Pro naše účely je potřeba navrhnut protokol, který bude spolehlivý, bude fungovat s dostatečným počtem síťových jednotek, ale vzhledem k dosahu technologie ZigBee, která v praxi je přibližně 50m, nemusí být toto číslo příliš velké, dále pak je kladen důraz na rychlosť a jednoduchosť.

Z těchto požadavků vyplývá, že je potřeba dbát ohled na propustnost sítě. Přenosová rychlosť standardu ZigBee je 250Kb/s a propustnost sítě se snižuje exponenciálně s rostoucím počtem jednotek a tedy i počtem vysílaných zpráv. Dále má vliv na propustnost i délka jednotlivých zpráv. V článku (Yang X., n.d.) je zpracována rozsáhlá studie na toto téma. Je zde použito vysílací schéma, které používá náhodnou dobu prodlení před odvysíláním zprávy a opakuje vysílání zprávy po definovaný časový úsek. Zároveň se do protokolu zabuduje mechanismus, který má za úkol co nejrychleji posílat kritické zprávy, to znamená, že se vysílání opakuje s velmi malou periodou a s rostoucím časem se pak interval zvyšuje. Díky této myšlence je pak velmi vysoká pravděpodobnost doručení všech zpráv na všechny cílové jednotky při dobré propustnosti přenášených dat v síti. V grafu na obrázku 2.3 je vidět jak se zvyšuje časové zpozdění, když zůstává konstantní vysílací doba (tečkováně) a když se časový interval mezi vysíláním zvyšuje (plná čára). Zároveň můžeme vidět, že komunikace je bez problému použitelná i pro 100 jednotek v síti, což je pro dosah cca 50 metrů dostatečná hodnota. Z těchto studií vychází i náš návrh bezdrátového komunikačního protokolu. Další zajímavé informace s reálnými měřeními lze najít v (Brown, n.d.).



Obrázek 2.3: Graf závislosti zpoždění zpráv na počtu jednotek v síti

2.3 Požadavky na komunikační protokol

Aby mohl celý systém dobře fungovat, je nutné správně navrhnout komunikaci mezi jednotlivými zařízeními a vypořádat se co nejlépe s chybami, které mohou nastat. Zařízení bude určitým způsobem vyhodnocovat rychlosť, zrychlení a směr vozu. Z hardwarového hlediska jsme se rozhodli pro GPS modul a akcelerometr. Pro jednoduchost jsme se prozatím omezili na rychlostní komunikace, kde se předpokládá plynulost jízdy a malé změny směru.

Základní funkce zařízení je včasné varování před překážkou na vozovce a nebezpečnými manévrovy při jízdě tak, aby šlo předejít nehodě. Z toho vychází předpoklad na okamžité zahájení vysílání při nebezpečné situaci a to po celou dobu trvání nebezpečí. Zároveň je potřeba vyslat větší počet zpráv, kvůli chybám v komunikaci, ale jedná se o bezdrátový přenos, takže kapacita kanálu je omezena. Proto je potřeba implementovat výše zmíněný mechanismus na snižování počtu přenášených zpráv.

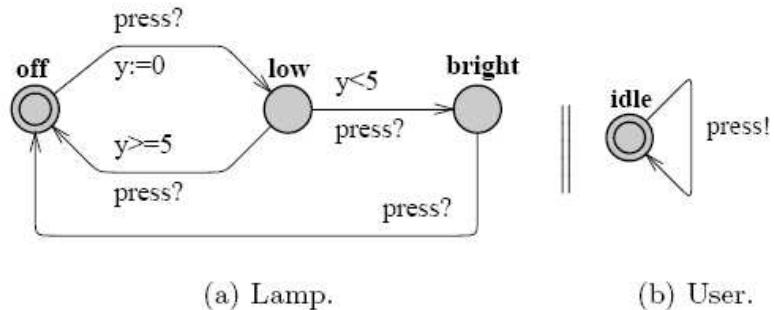
Dalším problémem je varování pouze vybrané skupiny automobilů. Systém tedy musí být schopen rozpoznat protijedoucí automobily, které nemají být varovány, ale zároveň mohou přepošílat varování dál. Posledním vhodným požadavkem je automatická aktivace

a deaktivace systému při vjezdu do zabezpečeného úseku.

2.4 Návrh protokolu

Komunikační protokol jsem navrhoval v prostředí *UPPAAL*, kde jsem následně provedl i jeho verifikaci. Tento nástroj slouží pro návrh sekvenčních automatů.

Sekvenční automaty (jinak také stavové automaty, časové automaty) jsou modely, které popisují chování systému a jeho časový vývoj. Velice jednoduše se tyto modely znázorňují graficky.



Obrázek 2.4: Příklad stavového automatu

Jak je vidět z předchozího obrázku 2.4 popisujícího závislost světelné lampy na mačkání spínače, skládají se automaty z uzlů a hran. Uzly reprezentují jednotlivé stavy, ve kterých se systém může nacházet, hrany pak přechody mezi nimi. Šipky pak určují směr přechodu. Systém se v jednom časovém okamžiku nachází pouze v jednom stavu, narozdíl od některých jiných typů grafických sítí (např. Petriho sítě). Přechod do dalšího stavu je většinou podmíněn logickým výrazem, který se cyklicky vyhodnocuje a tento přechod se uvolní jen při splnění podmínek výrazu. V případě, že přechod není uvolněn, zůstává automat v daném stavu. Počáteční stav, ve kterém se automat při spuštění nachází, je v prostředí *UPPALu* modelován dvojitým kolečkem. Od tohoto místa se pak dále vyvíjí.

Protože jednotlivé automaty mohou být časově provázány, existují zde tzv. synchronizační kanály, které mají za úkol při odpálení hrany v jednom modelu uvolnit zároveň hranu v modelu druhém. Na obrázku 2.4 je vidět jeden synchronizační kanál **press**, který spojuje modely a a b. Otazník u kanálu znamená, že model čeká na synchronizační pokyn,

vykřičník znamená, že kanál uvolňuje přechod pro jiný automat. Automat **b** reprezentuje tlačítko a při každém zmáčknutí se synchronizuje automat **a** představující světelný zdroj. Ten potom v závislosti na vyhodnocování logických podmínek se nachází ve stavu *Zhasnuto*, *Nízká úroveň osvětlení*, *Vysoká úroveň osvětlení*.

Prostředí *UPPAAL* slouží tedy pro návrh těchto sekvenčních automatů. Samotná práce v prostředí je velice intuitivní, ale je potřeba znát základní pojmy, které lze získat v dokumentaci (Behrmann, 2004). Druhou funkcí tohoto nástroje je možnost testovat správnost návrhu. K tomuto účelu slouží tzv. verifikační formule. Jsou to logické výrazy s definovanou syntaxí, které *UPPAAL* vyhodnotí a dá uživateli vědět, zda jsou pravdivé, či nikoliv. Tímto se způsobem se dá model testovat například na časové uvíznutí (dead-lock), přechod do konkrétního stavu nebo do vybraných stavů několika modelů a další. Podrobnosti o tomto nástroji lze nalézt na internetové stránce projektu www.uppaal.com.

Z důvodu složitosti schématu jsem byl nucen rozdělit celý návrh do dvou částí - vysílací část a přijímací část. Obě části se pak dále skládají z několika modelů, které jsou časově provázány a dohromady simulují požadovanou funkci.

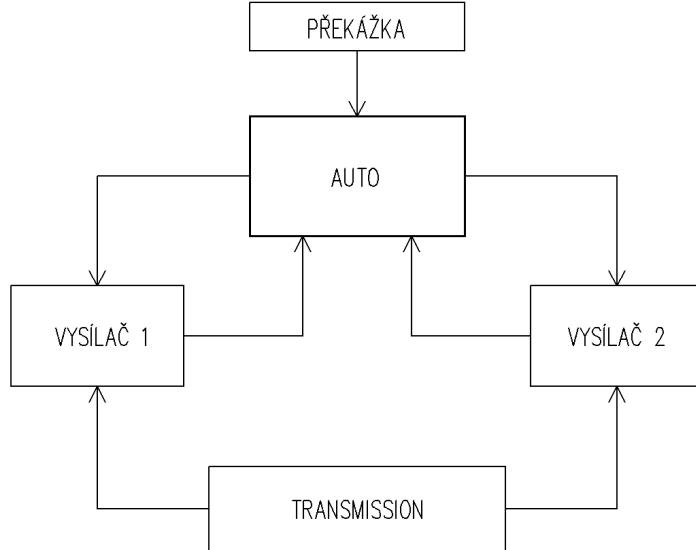
2.5 Vysílací část protokolu

2.5.1 Princip návrhu

Zařízení začne reagovat pokud se automobil dostane do abnormálního stavu. Pokud se tak stane začne ihned vysílat varovné zprávy EWM (Emergency Warning Message) s maximální frekvencí a postupně, aby uvolnil kanál, frekvenci vysílání snižuje. Vysílá tak dlouho dokud je vůz hrozbou pro své okolí. Druhou možností, kdy systém vysílá EWM je, když přijme varování od jiného vozu a je potřeba zprávu dále předat. To se děje kvůli preventivnímu varování vzdálenějších vozů nebo vozů, které díky rušení nemohly zachytit signál od iniciátora vysílání.

Aby byla zjednodušena komunikace mezi jednotlivými vozy a nějakým způsobem ošetřena chybovost přenosu, nekontroluje vysílač zda zprávu někdo přijal, ale vysílá EWM stále dokola s náhodnou dobou mezi jednotlivými přenosy. Odpadá tedy kontrola přijatých zpráv a díky systému "předávání" EWM se zajistí redundantní množství zpráv.

2.5.2 Podrobný popis funkce



Obrázek 2.5: Blokové schéma vysílače

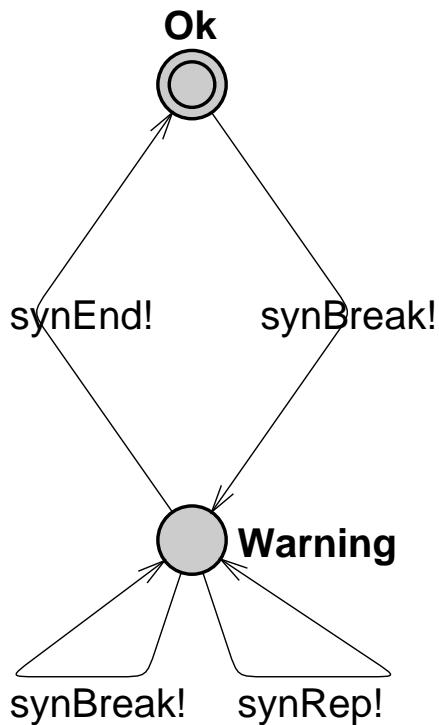
Na obrázku 2.5 je vidět blokové schéma vysílače. Jednotlivé bloky představují stavové automaty (podmodely) celého modelu. Základním předpokladem pro zahájení vysílání je událost od automatu *Překážka*, který má za úkol detektovat nebezpečnou situaci. Tuto informaci předá bloku *Auto*, které pak řídí celé vysílání a používá k tomu dva vysílače. První z nich vysílá zprávu s vysokou frekvencí a postupně tuto frekvenci snižuje. Po odvysílání určitého počtu zpráv uvědomí blok *Auto* a to začne vysílat pomocí druhého vysílače zprávy s nižší konstantní frekvencí. Při každém odvysílání zprávy uvědomí automat *Transmission* vysílače o dokončení tohoto úkonu. Model přestane vysílat, pokud blok *Překážka* vyhodnotí situaci jako bezpečnou. Na obrázku 2.5 je tedy vidět blokové schéma bezdrátového vysílače automobilu.

Celý návrh jsem se snažil co nejvíce zjednodušit a tím pádem použít i co nejméně časových proměnných, abych šetřil výpočetní výkon stroje. Definice globálních proměnných obsahuje pole kanálů - každý automobil má 10 synchronizačních kanálů, přes které jsou časově propojeny všechny části návrhu. Dále je zde vysílací vektor, který definuje dobu mezi jednotlivými zprávami. Ve stavových automatech *Vysílač1* a *Vysílač2* je pak ještě použita jedna proměnná *clock*, která omezuje přechody mezi stavů a proměnná typu *int*, která hlídá počet průchodů smyčkou. Model komunikačního protokolu jsem tedy rozdělil do následujících automatů:

- *Auto* - tento automat popisuje základní stavy automobilu. Přechody mezi jednotlivými stavy jsou synchronizovány s ostatními automaty.
- *Vysílač1* - model vysílače, který má proměnnou délku mezi jednotlivými vysílanými zprávami. Začíná vysílat s maximální frekvencí, která se postupem času snižuje. To se děje z důvodu požadavku okamžitého varování při nebezpečí a následném uvolňování kapacity přenosového kanálu.
- *Vysílač2* - tento vysílač slouží pro vysílání zpráv s konstantní frekvencí. Ta je samozřejmě nižší než u vysílače1, právě proto, aby příliš nezahlcoval kanál. To nastává buď po odvysílání celého vektoru Vysílače1 nebo při předávání zpráv (preventivní varování).
Oba vysílače byli původně zapouzdřeny do jednoho modelu, ovšem pro přehlednost a další zjednodušení byly rozděleny.
- *Překážka* - slouží pro detekci nebezpečného stavu a tím spustí veškeré vysílání varovných zpráv.
- *Transmission* - slouží pro detekci, že zpráva byla celá odvysílána.

Nyní podrobněji popíšu jednotlivá schémata a jejich stavy.

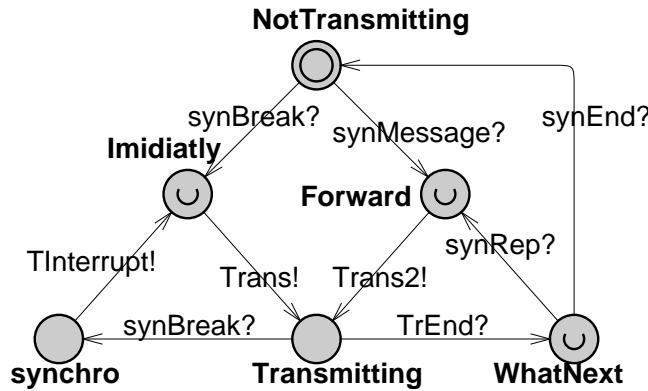
2.5.2.1 Překážka



Obrázek 2.6: schéma pro detekci nebezpečného chování

Slouží pro detekci nebezpečného chování. Pokud například řidič prudce zabrzdí přejde automat do stavu **Warning**, tím synchronizuje *Auto* a spustí vysílání. Ze stavu **Warning** může buď přejít zpět do stavu **OK** a tím ukončí vysílání nebo v tomto stavu zůstane a vysílání se opakuje přes synchronizační kanály **synRep** nebo **synBreak**.

2.5.2.2 Auto



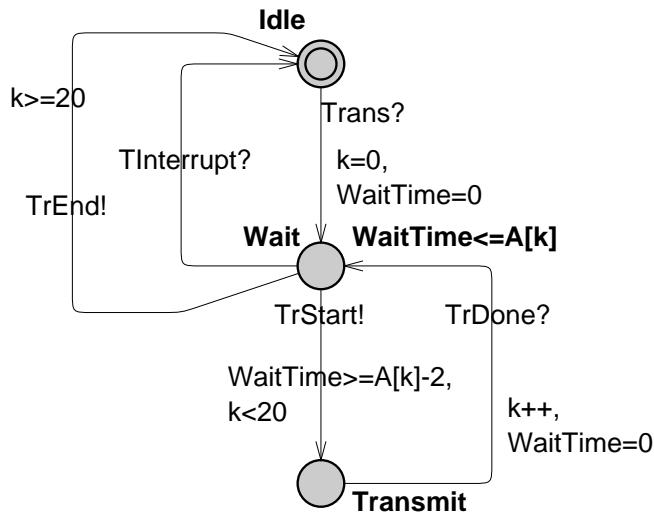
Obrázek 2.7: Model Auto

Toto je hlavní část návrhu, která rozhoduje ve kterém stavu se právě vysílač jako celek nachází. Zajišťuje základní funkci vysílání přes modely vysílačů. Má možnost určit, kdy bude který z dvojice vysílačů aktivní, zda-li se po odvysílání určitého počtu zpráv bude dále pokračovat nebo se automat vrátí opět do klidového stavu.

Základním stavem tohoto automatu je **NotTransmitting**, což je klidový stav a celý systém pouze poslouchá okolí a čeká na EWM. V případě, že řidič například sešlápně prudce brzdy, aktivuje se automat *Překážka*. Přes proměnnou **synBreak** synchronizuje model *Auto* a dostaneme se do dalšího stavu **Immediately**. Tento stav má za úkol spustit Vysílač1 a je typu **Urgent**. To znamená, že okamžitě (v simulačním prostředí s nulovým zpožděním) po příchodu do tohoto stavu se přesuneme do stavu následujícího a zahájíme vysílání. Po odvysílání se přes **TrEnd** dostaneme do stavu **WhatNext**, kde se rozhodne, zda-li má vozidlo stále abnormální chování - v tom případě se spustí *Vysílač2*, nebo nikoliv a přejdeme opět do klidového stavu.

Automobil může v tomto cyklu fyzicky již přestat být nebezpečným pro okolí, nicméně se dokončí vysílací cyklus (přibližně 2 sekundy) a až potom se ukončí vysílání. Během této doby ale může již dojít k dalšímu nebezpečnému manévrovi a proto je třeba obnovit opět maximální frekvenci vysílání. K tomu slouží opět synchronizační kanál **synBreak** ze stavu **Transmitting** a následně spustí opět *Vysílač1*.

2.5.2.3 Vysílač1

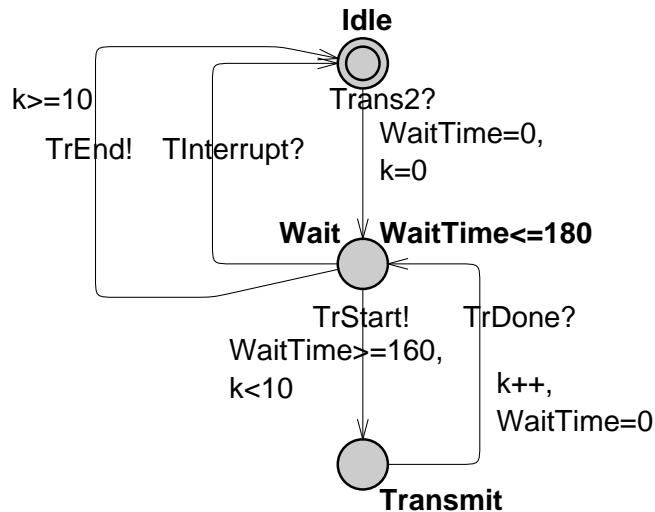


Obrázek 2.8: Model Vysilac1

Vysílač1 je model simulující posílání varovných zpráv přes bezdrátovou síť. Má dva základní stavů, které rozhodují, zda-li je aktivní, čí nikoliv a jeden mezistav, ve kterém čeká náhodnou dobu pro odvysílání další zprávy.

Vysílač čeká na synchronizaci od *Auto* a následně vynuluje proměnnou int *k* (tato celočíselná proměnná indikuje počet průchodů smyčkou a po dvacetí opakování vysílání ukončí) a proměnnou clock *WaitTime*, která nastavuje dobu čekání mezi jednotlivým odesíláním zpráv a přejde do stavu *Wait*. V tomhle stavu čeká definovanou dobu - podle vysílačního vektoru, který je zadán jako vstupní parametr a následně zahájí vysílání (pokud počet průchodů smyčkou je menší než dvacet). Tím synchronizuje *Transmission*, který hlídá odeslání zprávy. Po odeslání předá slovo zpět vysílači, který ukončí vysílání a přejde opět do stavu *Wait*. Cykly se opakují do doby než je počet průchodů větší než dvacet nebo se automobil znovu dostane do nebezpečného stavu (pomocí synchronizačního kanálu *TInterrupt*) a následně předá řízení schématu *Auto*.

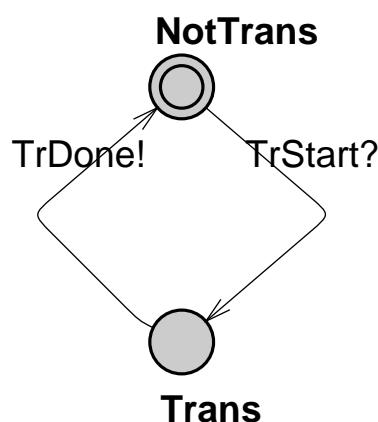
2.5.2.4 Vysílač2



Obrázek 2.9: Model Vysilac2

Tento automat je téměř shodný s *Vysílač1* pouze doba *WaitTime* je po celou dobu konstantní a počet průchodů smyčkou je zmenšen na deset, aby se snížila doba strávená ve smyčce.

2.5.2.5 Transmission



Obrázek 2.10: Model Transmission

Model slouží pro detekci odvysílání celé zprávy. Je spuštěn od vysílače, následně čeká dokud je zpráva celá odvysílána a opět předá řízení vysílači.

2.5.3 Verifikace

Po vytvoření všech komponent vysílací části jsem verifikoval systém pomocí *UPPAAL verifier* a testoval jeho vlastnosti. Zde jsou některé důležité verifikační dotazy a jejich formule.

1. Testování systému na uváznutí

$\text{A}[\cdot] \text{ not deadlock}$

Property is satisfied

Popis: testování systému na uváznutí (deadlock). Testujeme, zda-li se model nedostane do některého stavu, ze kterého se už nemůže vrátit. Pak by systém nemohl cyklicky vykonávat svou funkci.

2. Existuje stav, kdy vysílají např. tři vysílače současně?

$\text{E} <\!\!> \text{ Vx1.Transmit \&& Vx2.Transmit \&& Vx3.Transmit}$

Property is satisfied

Popis: dotaz ukazuje, že je možné, aby v jednu chvíli vysílalo více vozidel. Proto je nutné opakovat vysílání EWM a přidat před každý vysílání náhodný čas.

3. Je možné, aby vysílač začal znova vysílat aniž by dokončil předchozí cyklus?

$\text{E} <\!\!> \text{ A1.synchro \&& P1.Warning}$

Property is satisfied

Popis: automobil může začít generovat varovný signál a chvíli na to se chovat jako bezpečné vozidlo. Poté se ovšem dostane do nebezpečného stavu a je potřeba, aby vysílač začal opět vysílat s maximální rychlostí, což, jak je vidět, je splněno.

4. Je možné, aby zařízení vysílalo aniž by nenarušovalo bezpečný provoz?

E<> A2.Transmitting && P2.0k

Property is not satisfied

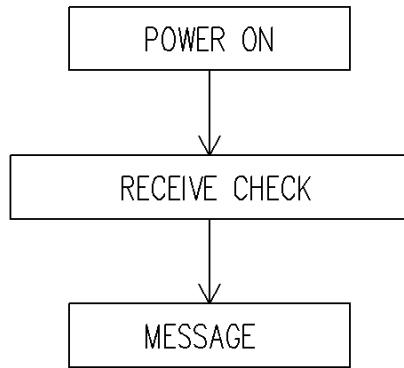
Popis: není to možné právě proto, že tento model neobsahuje přijímací část a systém tedy není schopen rozpoznat, zda-li přijal nějakou zprávu a mohl ji předat dál.

Další dotazy modelují různé situace a jejich formule jsou podobné, proto není nutno je zde všechny uvádět.

2.6 Přijímací část protokolu

2.6.1 Princip návrhu

Přijímací část komunikačního protokolu jsem také rozdělil na několik jednoduchých podautomatů, které jsou navzájem provázány. Tato část již ale není tak složitá jako vysílací schéma, protože celý proces se odehrává uvnitř zařízení. V případě vysílací části musíme brát v úvahu chování automobilu navenek - to jest, jakým způsobem ovlivňuje komunikaci mezi jednotlivými zařízeními. Všechny automobily mají vlastní vysílače, detektory překážek atd. a ty musí být jednoznačně přiřazeny jako jeden funkční celek. Oproti tomu model přijímače nikterak nezasahuje do vnějších záležitostí a pouze „poslouchá okolí“. Proto nemusím vytvářet složitou provázanost a můžeme tento model navrhnout zcela odděleně.

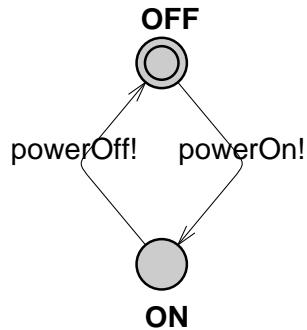


Obrázek 2.11: Blokové schéma přijímače

Na obrázku 2.11 je znázorněno blokové schéma přijímače. Automat *PowerOn* modeluje zapnutí celého systému a synchronizuje model *ReceiveCheck*, který přijímá jednotlivé zprávy a rozhoduje, zda-li jsou v pořádku. Pokud ano, předává je automatu *Message*. Ten má za úkol přijatá data zpracovat.

2.6.2 Podrobný popis funkce

2.6.2.1 PowerOn

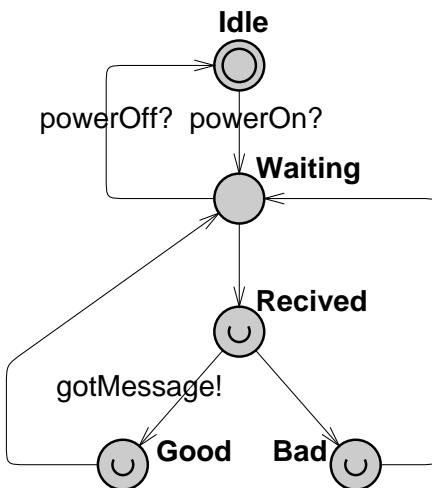


Obrázek 2.12: Model PowerOn

Tento jednoduchý automat uvádí celé zařízení do chodu. Jak bylo popsáno v předchozí části, celý systém je funkční v omezeném rozsahu - tedy například dálnice nebo pouze zabezpečený úsek komunikace. Při použití GPS lze například automaticky při nájezdu na dálnici uvést systém do chodu a předat mu informace týkající se polohy.

Při příchodu takového signálu přejde tento automat do stavu ON a přes synchronizační kanál powerOn zapne přijímač. Při příchodu vypínacího signálu přejde automat zpět do stavu OFF a synchronizuje opět schéma přijímače, čímž se celé zařízení vypne.

2.6.2.2 *ReceiveCheck*

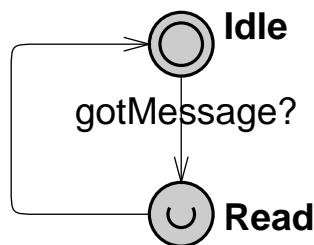


Obrázek 2.13: Model ReceiveCheck

Uvažujeme zde jednoduchý automat, který po zapnutí začne přijímat zprávy od okolí a nijak toto okolí neovlivňuje. Po přijetí dat zkонтroluje, zda-li zpráva byla přijata celá a obsahuje požadované informace, pokud ano, přečte ji a opět přijímá další zprávy, pokud nebyla dobře přijata, zprávu smaže a čeká na novou.

2.6.2.3 *Message*

Tento automat má za úkol po přijetí zprávy přejít do stavu READ, aby umožnil vyprázdnění komunikačního bufferu. Automat je synchronizován kanálem gotMessage automatu *ReceiveCheck*, který tuto synchronizaci umožní po příchodu správného formátu zprávy.



Obrázek 2.14: Model Message

2.6.3 Verifikace

1. Testování systému na uváznutí

`A[] not deadlock`

`Property is satisfied`

Popis: testování systému na uváznutí (deadlock) jako v případě modelu vysílače. Systém neuvázne.

2. Existuje stav, kdy přijímač dostal zprávu a ta je přečtena?

`E<> RCh.Good && M.Read`

`Property is satisfied`

Popis: dotaz ukazuje, že je možno přijmout zprávu a načíst informace, které obsahuje.

3. Je možné, aby systém zpracovával přijatou zprávu a byla přijata nová?

`E<> RCh.Received && M.Read`

`Property is satisfied`

Popis: je vidět, že pokud je přijata zpráva a ta je následně zpracovávána, může systém dále přijímat data.

4. Je možné, aby zařízení přečetlo přijatou zprávu, ikdyž nebyla přijata správně?

M.Read --> RCh.Received

Property is not satisfied

Popis: není možné, protože schéma *Message* čeká na pokyn k přečtení zprávy.

5. Může zařízení přijímat zprávy pokud je vypnuté?

E<> P.OFF && RCh.Waiting

Property is not satisfied

Popis: zařízení nemůže pracovat dokud nebylo aktivováno.

2.7 Zhodnocení

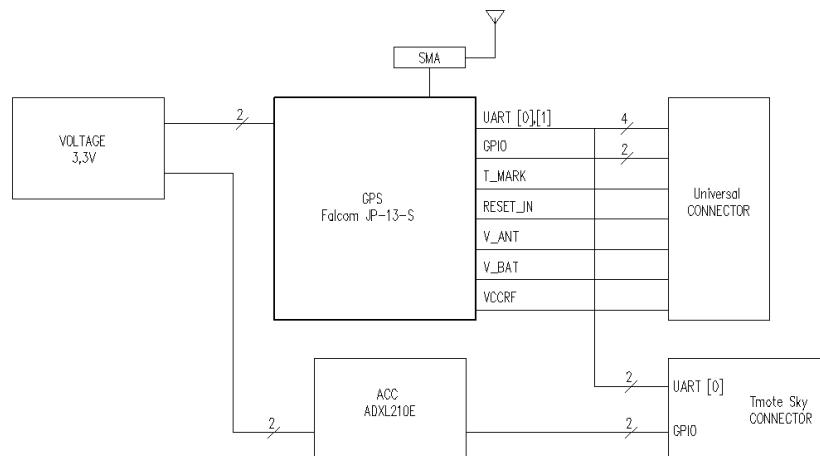
Navrhované části - vysílací model komunikačního protokolu a přijímací model komunikačního protokolu - jsou maximálně zjednodušeny kvůli minimalizaci stavového prostoru a jsou zverifikovány. Obě části byly testovány zvlášt', protože je potřeba velkého výpočetního výkonu pro testování všech stavů a odhalené chyby byly opraveny. Dalším krokem testování je vyzkoušet komunikaci na reálných zařízeních.

Kapitola 3

Hardware

3.1 GPS + ACC Board

3.1.1 Základní popis



Obrázek 3.1: Blokové schema navrženého modulu

Tato deska byla vyvinuta pro získávání základních údajů o poloze vozidla. První koncept nepočítal s použitím technologie GPS, později se ale ukázalo, že tato varianta je výhodnější, protože není závislá na dalších systémech a jednotkách - například na datech vysílaných rozmístěnými stanicemi. Původně se jednalo o minimalizaci výrobních nákladů

jednoho zařízení, které se s použitím GPS zvyšuje, ale ve skutečnosti se tím významně sníží náklady na vybudování celého systému a výrazně se zjednoduší komunikace v síti. Tím se stane celý systém i více robustní a není potřeba vytvářet další infrastrukturu pro správnou funkčnost. Na obrázku 3.1 je zobrazeno blokové schéma navrženého modulu. Modul se skládá ze dvou hlavních jednotek a to akcelerometru a GPS modulu. Všechny důležité signály jsou pak vyvedeny na univerzální konektor (ve schématu J2). Konektor Tmote Sky (ve schématu J1) slouží pro připojení na bezdrátový řídící modul Tmote-Sky a rozložení pinů J1 odpovídá rozložení na rozšiřujícím konektoru desky Tmote. Tyto dvě desky lze tedy jednoduše spojit a přenášet tak data z GPS (přes rozhraní UART) a akcelerometru (digitální vstupy/výstupy). Na tento konektor je ještě vyveden signál T_MARK z GPS modulu, což je přesný hodinový signál o frekvenci 1 Hz. To je možno využít například pro synchronizaci.

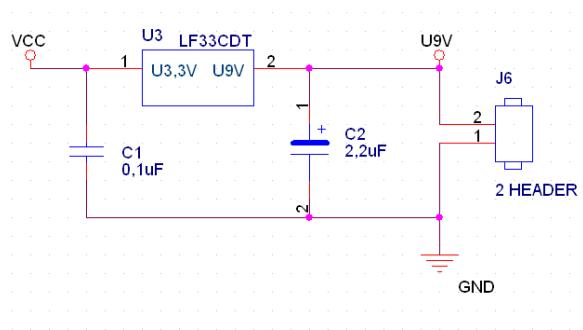
Pro určení polohy byl zvolen GPS modul firmy Falcom JP13-S, který používá již třetí generaci chipu SirfStarIII a podporuje až 20 přijímacích kanálů. Je integrován v jednom pouzdře v BGA provedení. Podrobnosti naleznete v (Fal, 2006). Ke zpracování změrených údajů slouží dva sériové kanály pro přenos do nadřazené jednotky. První kanál (port A) slouží pro klasická data z GPS (Global Position System), zatímco druhý kanál (port B) se používá pro DGPS (Diferencial GPS). Při využití jednoho kanálu je udávaná nepřesnost měřené polohy do 5 m. To je zaviněno především průchodem signálu přes atmosféru a jeho zakřivením (lom signálu při přechodu na různých vrstvách). Dalšími ovlivňujícími faktory jsou potom kvalita signálu, počet družic v dosahu, zpoždění signálu. Při druhém způsobu měření - DGPS, se využívá obou sériových kanálů, přičemž hlavní seriový kanál slouží na příjem konfiguračních zpráv a vysílání změřené polohy a druhý kanál potom přijímá data od objektu s přesně určenou polohou. Ten také přijímá družicový signál a vypočítává jeho chybu ze známé polohy. Tu pak vysílá ostatním přijímačům. V tomto módu měření jsou pak výsledky daleko přesnější. Hodnota přesnosti není uvedena v datasheetu, protože aktuální verze firmware modulu nepodporuje DGPS. Nevýhodou toho způsobu měření je nutnost vybudování rozsáhlé pozemní infrastruktury stanic s danou polohou a proto se tohoto způsobu u běžných aplikací nevyužívá. V mnoha případech k tomu také není ani důvod. Napěťové úrovně signálů na obou portech jsou CMOS, což je vyhovující pro komunikaci s řídící jednotkou, ale pro komunikaci s počítačem je třeba zařadit mezi obě zařízení napěťový převodník, například MAX3232.

Dalším prvkem pro určení relativní polohy, zvláště v kritických situacích, je akcelerometr. Pomocí něj lze v síti zjistit, které vozy přejíždějí mezi pruhy, brzdí nebo dělají jiné důležité manévry, na které je třeba dát pozor. Zároveň lze pak určovat polohu mezi jednotlivými

měřeními z GPS. Použitým akcelerometrem je dvouosý měřič zrychlení ADXL210AE od firmy Analog Devices, měřící zrychlení do 10 g. Podrobnosti v datasheetu, viz (Analog, 2002). Při nárazu automobilů dochází ke zrychlením mnohonásobně větším (40 - 150 g), což je ovšem již ve fázi nárazu a tedy z našeho hlediska pozdě. Pro naše účely plně postačuje detekce hodnot v rozsahu do 10 g, kdy jsme schopni situaci vyhodnotit jako nebezpečnou.

3.1.2 Napájení

Nominální napětí pro JP13-S je 3,3 V, což lze použít i pro akcelerometr. Napájecí napětí je tedy řešeno pro celou desku společně. Je zde použit stabilizátor LF33CDT s výstupním napětím 3,3 V a vstupním napětím 5 až 15 V. Můžeme tedy využít i napájecí rozvody v automobilech. Stabilizátor napětí potřebuje ke své funkci dále dvojici kondenzátorů, a to C1 a C2, které jsou umístěny co nejbliže, protože se jedná o filtrační kondenzátory. Zemnící pin stabilizátoru je propojen pomocí návěstí a není v schématu na obrázku 5.2 vidět. Pro účel testování byla deska osazena držákem na 9V baterii.

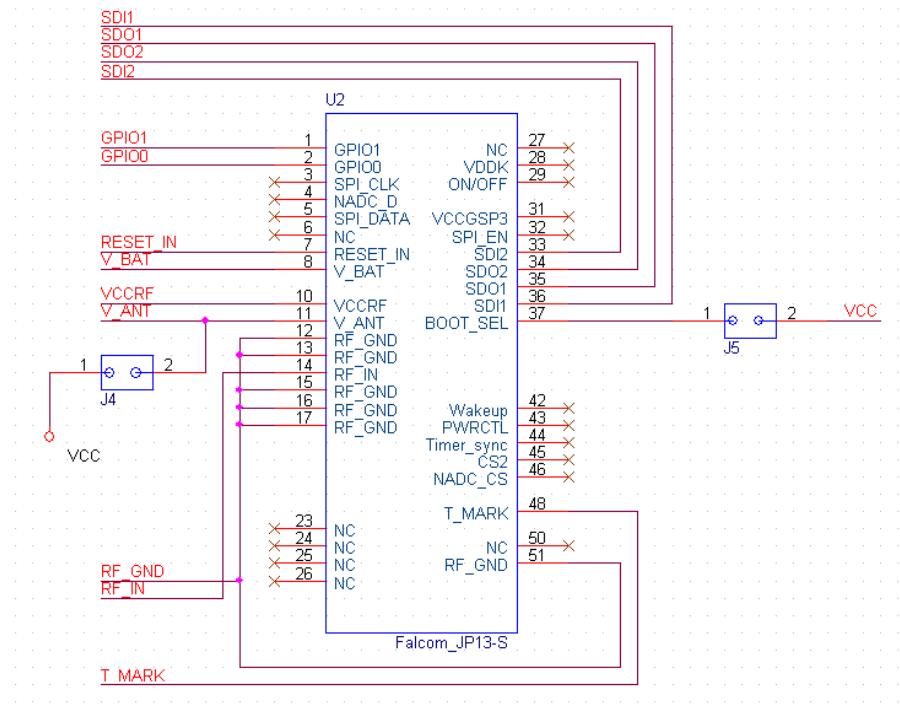


Obrázek 3.2: Napájení navržené desky

GPS modul, zobrazen schematicky na obrázku 3.3, má několik napájecích pinů VCC a GND, které jsou připojeny k napájecí sběrnici. Druhým typem zemnících pinů jsou RFGND, což je nulová napěťová úroveň pro radiový signál. Ty jsou také propojeny a vyvedeny na anténní zem. Obě země jsou propojeny interně a nesmí být spojeny v žádném jiném bodě. Dalším napájecím vstupem je zde pin V_ANT, který slouží pro napájení aktivní antény. Hodnota napětí nesmí přesáhnout 12V a záleží na použité anténě. Tento pin je na desce připojen přes propojku J4 na napájecí napětí 3,3V. Po rozpojení J4 můžeme připojit externí zdroj přes pin 6 konektoru J2. Modul navíc obsahuje výstupní

pin VCCRF, který může sloužit také pro napájení antény a je na něm vyvedeno napětí 2,85 V.

3.1.3 Popis desky plošného spoje

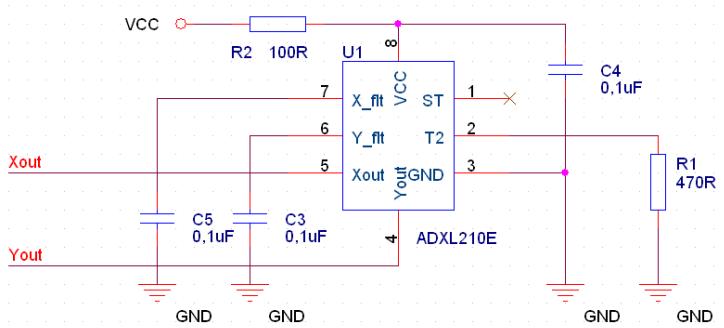


Obrázek 3.3: Schéma připojení GPS modulu

Z výstupu stabilizátoru jde napětí na dvojici propojek J4, J5. Jak již bylo řečeno, J4 slouží pro připojení VCC pro napájení antény GPS a J5 při propojení nastaví pin BOOT_SELECT do stavu "High", čímž nastaví chip do režimu pro update nové verze firmware.

Na konektor J1 je vyveden seriový kanál 1 GPS modulu pro komunikaci s řídícím modulem a signál T_MARK využitelný pro časovou synchronizaci. Na konektor J2 jsou vyvedeny stejné piny jako na J1 dále pak druhý port 2 seriového kanálu GPS, dva vstupy/výstupy GPIO0, GPIO1 (využití pro budoucí verzi firmware), signál RESET_IN pro resetování přijímače, VCCRF sloužící pro napájení rádiové části, V_ANT pro napájení aktivní antény GPS a nakonec pin V_BAT jako záložní zdroj energie při výpadku napětí na VCC. Hodnota tohoto napětí musí být 3V. Pokud záložní baterii nepoužíváme lze

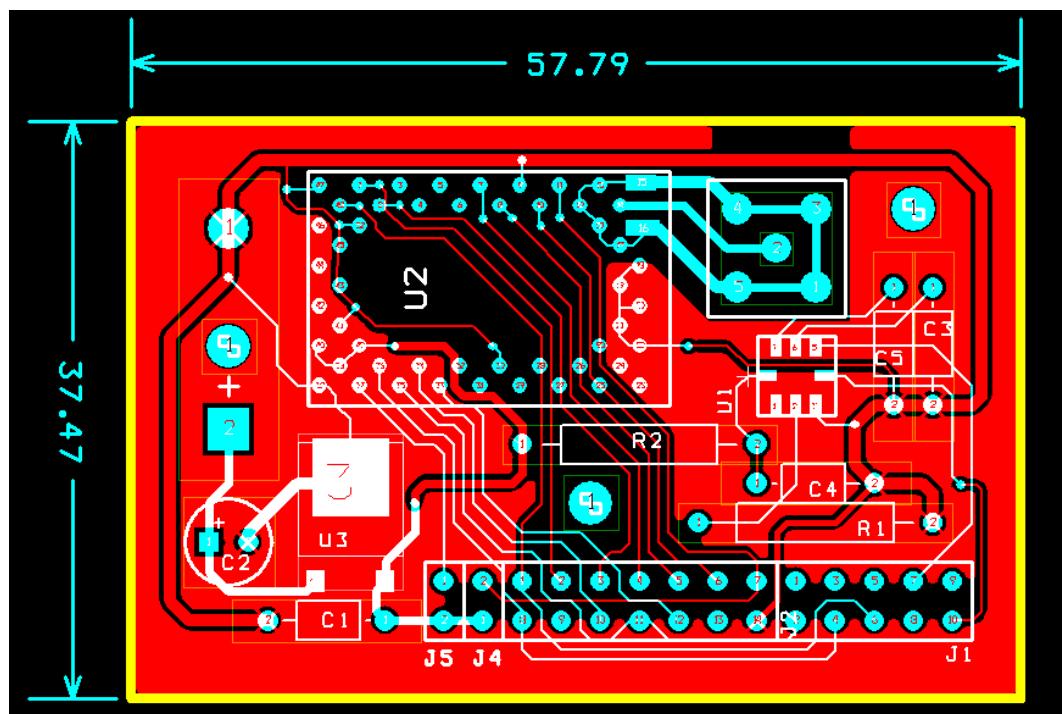
tento pin ponechat nezapojen nebo se připojí na zem.



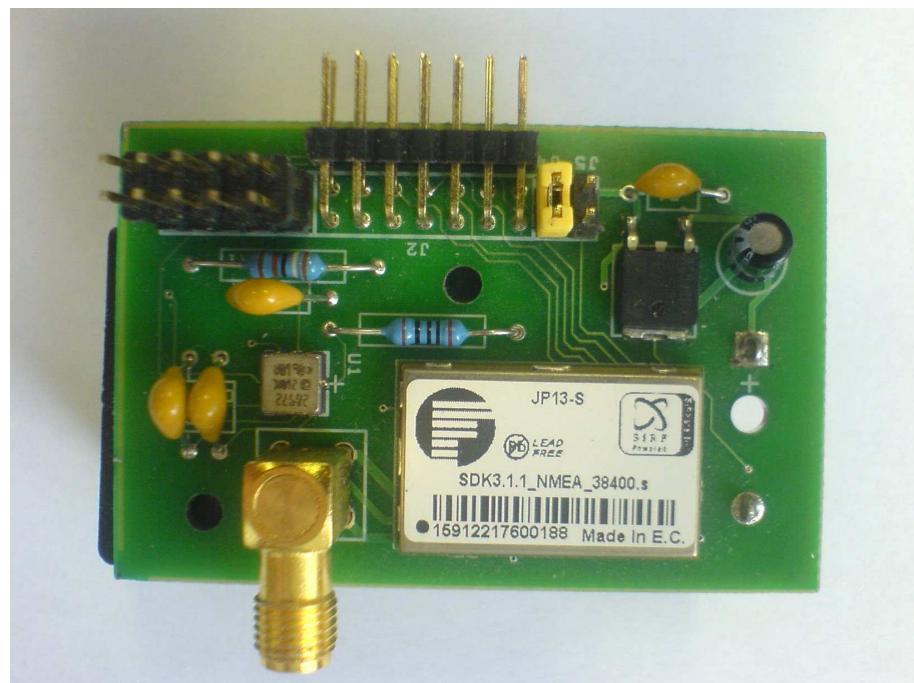
Obrázek 3.4: Schéma zapojení akcelerometru

Na obrázku 3.4 je zobrazeno zapojení akcelerometru. Pro svou funkci potřebuje několik externích součástek. Na vstupech X_{FILT} a Y_{FILT} jsou připojeny kondenzátory C3 a C5. Chovající se jako dolnofrekvenční propust spolu s interním rezistorem R_{FILT} zabraňující antialiasingu a snižují šum. Rezistor R1 slouží pro nastavení periody výstupních signálů akcelerometru a konečně R2 spolu s C4 slouží jako filtrační člen pro napájecí napětí. Hodnoty součástek lze vypočítat dle vzorců uvedených v datasheetu akcelerometru (Analog, 2002).

Jako anténu k GPS jsem použil anténu FAL-ANT-3 od firmy Falcom. Jedná se o aktivní anténu s magnetickou podložkou a je určena k namontování na kovový povrch. Je zakončena konektorem typu SMA-male a na desce je proto osazen konektor SMA-female jako J3. Navržený plošný spoj je dvouvrstvý a má rozměry 58 x 37 mm. Je navržen v páté konstrukční třídě přesnosti pomocí systému OrCAD. Rozměry odpovídá přibližně velikosti modulu Tmote, na který se dá napojit jako rozšiřující deska. Navržený plošný spoj je na obrázku 3.5, vyrobená a osazená deska pak na obrázku 3.6.



Obrázek 3.5: Navržený plošný spoj desky



Obrázek 3.6: Vyrobená deska GPS/ACC

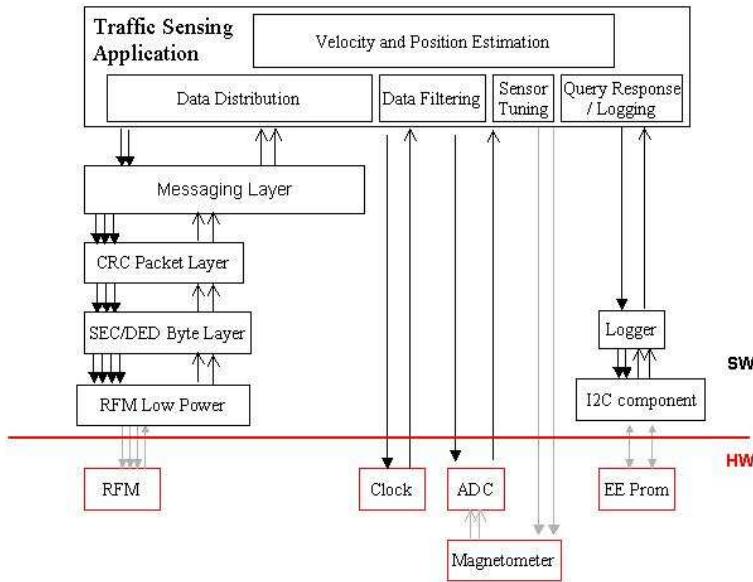
Kapitola 4

Software

4.1 TinyOS

4.1.1 Úvod

TinyOs je operační systém s open-source licencí navrhnutý pro bezdrátové senzorové sítě. Používá programovací jazyk nesC, což je nadstavba jazyka C a umožňuje lépe implementovat výhody tohoto operačního systému. Pro tuto platformu bylo navrženo několik hardwarových modulů a to jak komerčních, tak výukových, u nichž jsou běžně k dispozici jejich elektrická schémata. V dnešní době je na trhu přibližně deset těchto zařízení, naše katedra zakoupila několik kusů modulu T-mote Sky, který byl původně vyvinut na Univerzitě v Berkeley a nyní jej distribuuje společnost MoteIV.

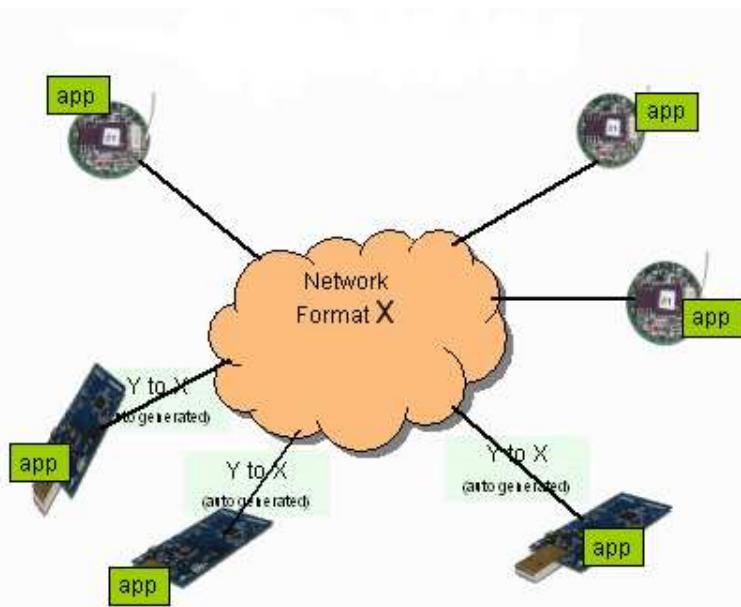


Obrázek 4.1: struktura programování pod TinyOS

Na obrázku 4.1 je vidět struktura programování pod TinyOS. Každá hardwarová komponenta má svou softwarovou komponentu, která s ní komunikuje na nejnižší softwarové úrovni ISO/OSI modelu a zprostředkovává vazby mezi vyššími vrstvami. V některých případech je pro daný modul implementováno více vrstev, které jsou využívány podle potřeby vývojáře, například radiový přenos dat. Nebo v případě seriové komunikace lze na nejnižší vrstvě přenášet jednotlivé Byty, na vyšší vrstvě to je speciální packet a na nejvyšší úrovni probíhá packetová komunikace s potvrzováním příjmu. Podle typu aplikace se lze nalinkovat na kteroukoliv vrstvu a ta pak sama komunikuje s vrstvami na nižší úrovni.

Jelikož TinyOS je otevřenou platformou a každý má možnost zúčastnit se vývoje a podpory různých hardwarových komponent pro tento operační systém, je na oficiálních stránkách k dispozici značné množství souborů s dokumentací, tzv. TEP soubory, které jsou rozdeleny podle hardwarových bloků. Tedy například soubory popisující bezdrátový radio modul, sériovou komunikaci, časovače a čítače atd. Zde jsou částečně popsány rozhraní poskytující ovládání k těmto blokům a doporučení pro vývojáře, jak by měla vypadat struktura těchto rozhraní a jaké ovládací prvky by měla obsahovat. Záleží pak na každém výrobci hardwarového modulu, jak se těchto doporučení bude držet a jak kvalitně své softwarové prostředky zdokumentuje. Na obrázku 4.2 je znázornění komunikace mezi

moduly postavenými na různých platformách. To je možné díky použití síťových hardwarově nezávislých typů proměnných, kdy překladač pro daný procesor zařídí, že hodnoty proměnných budou správně reprezentovány. Tím se zajistí bezproblémová komunikace a částečná přenositelnost kódu.



Obrázek 4.2: Bezdrátové moduly pro TinyOS

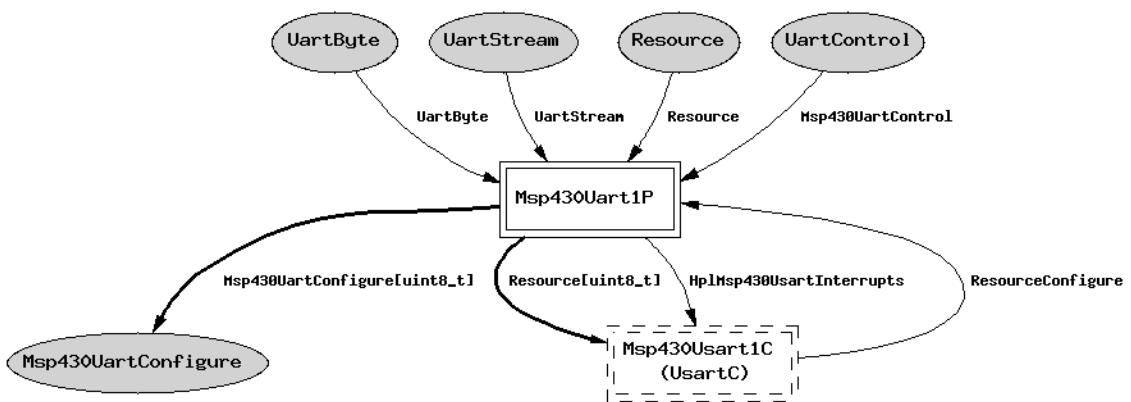
Základní myšlenka TinyOs je rozdělení kódu na jednotlivé komponenty, což umožňuje přehlednější strukturu celého programu a zároveň zmenšuje velikost celé aplikace, protože lze provozovat tyto komponenty dohromady a tím i sestavit nároky na paměťový prostor u těchto senzorových sítí. Součástí tohoto operačního systému je velké množství knihoven a komponent, které zahrnují komunikační protokoly SPI, I2C, bezdrátová komunikace, asynchronní sériový přenos a další. Dále pak knihovny obsluhující senzory těchto modulů, převodníky, časovače, rozšiřující vstupy a výstupy a jiné hardwarové prostředky. Posledním hlavním prvkem knihoven tohoto operačního systému je soubor nástrojů usnadňujících sběr a distribuci změrených dat, které pak mohou být jednoduše využity ve vlastních aplikacích.

TinyOS je systém řízený událostmi, což znamená, že veškeré prováděné akce se dějí na základě vzniklého hardwarového přerušení. Výhodou této struktury je jednodušší plánování operací vykonávaných v procesoru a přispívá to k již zmíněnému zpřehlednění uživatelského programu, naopak nevýhodou systémů s přerušením je nutnost vhodně spravovat veškeré události tak, aby nedocházelo k časově neefektivním prodlevám a

případným zacyklením programu vinou špatné implementace. K tomu ale jazyk nesC nabízí určité nástroje, které mají vývojáři pomoci. Tím můžou být atomické formule, ve kterých nemůže dojít k přerušení, vytváření úloh (*task*), prováděných až při nevytížení procesoru atd. Pro bližší seznámení s těmito pokročilejšími prvky programování odkazují na (Levis, 2006).

TinyOS je portován již na více než deset hardwarových platform a několik senzorových desek. Je používán širokou komunitou uživatelů, zatím především k výzkumným účelům na univerzitách a vědeckých institucích převážně v zahraničí. U nás na katedře se s těmito bezdrátovými technologiemi zabývá několik projektů popsaných na katedrálním serveru rtime.felk.cvut.cz v sekci HW. Při vývoji vlastního hardwaru pro ZigBee komunikaci by mělo být snahou podporovat tento již poměrně rozšířený operační systém a vytvořit softwarové moduly začlenitelné do struktury tohoto operačního systému.

4.1.2 Komponenty



Obrázek 4.3: Provázání komponent v konfiguračním souboru

Jak již jsem zmínil celý kód aplikace pro TinyOS se stává z jednotlivých komponent. Existuje zde jistá podobnost s ostatními programovacími jazyky, kde se například odkazujeme na jednotlivé metody a proměnné jiného objektu přes zadání jména objektu vlastníčího tyto prvky. Stejně tak musíme provázat i metody v jazyce nesC, ale tento krok je trochu odlišný. TinyOS spolu s nesC využívá k definici metod jednotlivých modulů rozhraní neboli interface (dále jen rozhraní). Tyto rozhraní uchovávají seznam metod a událostí, které mohou být použity a musí se dále vázat na soubor s implementací těchto prvků.

K tomuto účelu slouží komponenta typu konfigurační soubor, která má za úkol propojit všechny soubory tvořící aplikaci a zároveň dává informaci o tom, ke kterému modulu dané rozhraní patří. Třetím důležitým typem souboru, který TinyOS využívá, je komponenta zvaná modul. Zde se nachází uživatelský kód, implementace všech rozhraní a knihoven tohoto operačního systému. Každý modul musí mít uveden v příslušné sekci seznam rozhraní, které chce využívat a které sám poskytuje. Tedy například:

```
module TestUartOP {
    provides {
        interface StdControl;
        interface TestUartFeedback;
    }
    uses {
        interface ResourceCmd as ResourceDebug;
        interface ResourceCmd as ResourceSend;
        interface Resource as ResourceReceive;
        interface HPLUSARTControl as UartControl;
        interface HPLUSARTFeedback as UartFeedback;
        interface Timer2<TMilli> as Timer;
        interface Timer2<TMilli> as TimerConflict;
        interface Leds;
        interface StdControl as SubControl;
    }
}
```

Rozhraní jsou tedy jakýmsi prostředníky mezi jednotlivými částmi a pokud tedy chceme použít například časovač nebo LED diody našeho hardwarového modulu použijeme rozhraní Timer2 a Leds, která poskytují metody, funkce a události pro ovládání těchto prvků.

V tomto příkladě je také vidět použití parametrických rozhraní Timer2<TMilli>, kde za jeho název uvádíme předávané parametry. V tomto případě uvádíme, že se jedná o čítání času v milisekundách. Některé komponenty jsou tzv. generické komponenty. Tato vlastnost je implementovaná až od verze 1.2 TinyOs. V normálním případě pokud vážeme spolu jednotlivé prvky kódu, nejsou vytvářeny jejich instance. Tento způsob je podobný například v C++ použití statické třídy a znamená to, že všechny části se odkazují na stejný kus kódu. V případě generických komponent vytváří překladač instance těchto

prvků, kde každá má v paměti svůj vlastní prostor. Tím samozřejmě narůstají paměťové nároky, ale například v případě časovačů je potřeba vytvořit v paměti vlastní prostor pro vícero časování. Instance generických komponent se vytvářejí klíčovým slovem *new* v sekci výčtu komponent v konfiguračním souboru. Za klíčovým slovem *as* je pak uvedeno jméno proměnné odkazující na tento objekt. Měnit jména užívaných rozhraní je možné i v případě klasických komponent. Dříve se místo generických komponent používalo parametrických rozhraní, kdy parametr určoval příslušný paměťový prostor (můžeme si představit jako instanci) komponenty. To je přístup, který můžeme ještě najít u starších komponent. Například rozhraní *GenericCom* pro radiovou komunikaci.

```
configuration TestUart0 { } implementation { components
    new MSP430ResourceUART0C() as ResourceCmd,
    new MSP430ResourceUART0C() as ResourceCmd1,
    new MSP430ResourceUART0C() as ResourceCmd2,
    LedsC,
    new TimerMilliC(),
    new TimerMilliC() as TimerConflictC,
    GenericComm as comm;
```

Momentálně tedy máme vytvořeno několik rozhraní, které chceme používat v našem programu a zároveň máme ošetřeno vytvoření instancí nebo použití statických komponent. Ted' zbývá pouze správně provázat rozhraní s jejich implementací. To se děje také v konfiguračním souboru. K tomuto účelu slouží tři operátory „ \rightarrow “, „ \leftarrow “ a „ $=$ “. První dva slouží pro základní navázání komponent, šipka vyjadřuje závislost od uživatele k poskytovateli. To znamená, využíváme-li v naší komponentě *TestUart0P* dva časovače *TimerMillic()* musíme je spojit s komponentou, která toto rozhraní poskytuje. Tedy:

```
TestUart0P.Timer -> TimerMilliC;
TestUart0P.TimerConflict->TimerConflictC;
```

Pro odkazování na jména rozhraní se používá tečková konvence. Pokud bychom byli důslední, měli bychom psát:

```
TestUart0P.Timer -> TimerMilliC.Timer2<TMilli>;
TestUart0P.TimerConflict -> TimerConflictC.Timer2<TMilli>;
```

Překladač TinyOs ale již ví, že naše proměnná Timer a TimerConflict je instance rozhraní Timer2<TMilli> a v komponentě TimerMilliC sama přiřadí správnou implementaci rozhraní a jeho metod. Komponenta TimerConflictC vznikla pojmenováním vytvořené instance TimerMilliC. Třetí operátor „=“ sděluje překladači, že dané rozhraní má svou vlastní implementaci v přiřazovaném modulu. To znamená, že kód rozhraní, stojícího na levé straně rovnítka, lze nalézt v modulu na pravé straně. Toho se používá nejčastěji v komponentách, které samy poskytují rozhraní a odkazují tak na implementaci ve svém vlastním modulu.

Na obrázku 4.3 je vidět grafické znázornění konfiguračního souboru Msp430Uart1C. Tato komponenta poskytuje čtyři rozhraní, která jsou na obrázku nahoře a váže jejich implementaci do modulu Msp430Uart1P. Tento modul využívá (pomocí klíčového slova uses) rozhraní Msp430UartConfigure, které má jako parametr osmibitový integer. Dále se váže na modul MSP430Usart1C, což je generická komponenta. Rozdíl mezi těmito dvěma přístupy je v tom, že rozhraní z komponenty MSP430Usart1C jsou využívána pro vnitřní funkci tohoto modulu a další komponenty se s nimi přímo nedostanou do styku. Kdežto v případě rozhraní MSP430UartConfigure modul implementuje jeho události a tato implementace je pak platná pro všechny komponenty, které tento modul používají.

To jak k jednotlivým modulům a jejich rozhraním přistupujeme se rozlišuje v konfiguračním souboru. Pokud je použit operátor „→“ nebo „←“ pro provázání daného modulu, znamená to použití pro vnitřní funkce, operátor „=“ uvozuje vlastní implementaci.

Ještě je třeba podotknout, že pro zachování přehlednosti se jednotlivé soubory značí odlišně podle jejich funkce, i když to nemusí být pravidlem. Konfigurační soubor mívá na konci názvu písmeno C, což značí *configuration*, modul bývá značen písmenem M (Module) nebo P (Program) a rozhraní je bez označení. Pro další podrobnosti o programování v TinyOS odkazují na (Levis, 2006), jako příklad poslouží okomentovaný kód, který je přiložen k této práci.

4.1.3 Konfigurace TinyOS

Programování pod TinyOs lze provozovat jak v operačním systému Windows, tak i pod Linuxem. Pro programování ve Windows je potřeba nainstalovat „napodobeninu“ linuxové příkazové řádky a příslušné softwarové balíčky podporující toto prostředí. Obecně se vývojem těchto nástrojů zabývá projekt *Cygwin*. Jeho domovská stránka je na internetové adrese www.cygwin.com, kde jsou volně k dispozici instalační balíčky a zdrojové kódy spolu s dokumentací.

V instalaci TinyOS je zahrnut i softwarový balík Cygwinu ve starší verzi. V případě potřeby (některé nástroje jako například debugger procesoru MSP430 potřebují pro svou funkci novější balík) lze stáhnout novou a přeinstalovat stávající verzi Cygwinu.

Dalšími důležitými prvky pro správnou funkci operačního systému TinyOS jsou soubory knihoven, nástrojů, rozhraní, překladačů atd. Naše instalace TinyOS obsahuje tyto základní balíčky:

```
tinyos-tools-1.2.2.1
msp430tools-binutils-2.17-20060802
make-3.80tinyos-1 tinyos-javacomm-1.0.0-1
moteiv-tools-0.1-20060808
nesc-1.2.7a-1
msp430tools-base-0.1-20050607
msp430tools-gcc-3.2.3-20060802
msp430tools-libc-20060208cvs-20060208
tinyos-1.1.15Dec2005cvs-1
tinyos-moteiv-2.0.4-1
```

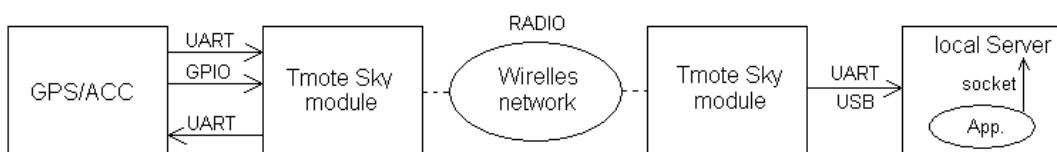
Zde je nutno sdělit, že TinyOS je vyvíjen ve dvou různých, ale podobných, větvích. Obě z nich jsou nekomerční a lze je tedy bezplatně užívat. Jednou z nich je větev distribuovaná uskupením *TinyOS Alliance* a můžeme zde najít podporu pro téměř všechny bezdrátové moduly, druhou větví je pak verze *Boomerang*, kterou distribuuje společnost *MoteIV* a ta zahrnuje podporu pro své vlastní produkty, mezi nimiž najdeme i moduly Tmote Sky používané na katedře řídící techniky.

V současnosti jsou dostupné tři verze TinyOS a to starší TinyOS 1.1, která se stále zachovává kvůli kompatibilitě (ve verzi 2.0 byly předělány některé komponenty, které nelze ve staré verzi použít a naopak), verze TinyOS 2.0 a nakonec Boomerang 2.0.4. Jak je vidět z předchozího výpisu balíčků, používám Boomerang 2.0.4, která je ovšem postavena na verzi TinyOS 1.1. Protože moduly Tmote Sky jsou osazeny procesorem MSP430 od Texas Instruments, je implementovaná podpora pro tento typ i s překladačem gcc pro tento procesor, dále pak podpora Javy, podpora jazyka nesC a nástroje klasického TinyOS ve verzi 1.2, což už zahrnuje například i výše zmíněnou podporu generických komponent a nakonec balíček nástrojů od MoteIV. Další podrobnosti a odlišnosti v jednotlivých verzích lze nalézt na internetové adrese [urlwww.tinyos.net](http://www.tinyos.net).

Jako vývojové prostředí pro tvorbu aplikací lze použít, jak bývá zvykem pro Unixové systémy, libovolný textový editor. Nicméně pro zachování přehlednosti a ulehčení kódování je dobré mít přinejmenším zvýrazněnou syntaxi jazyka *nesC*. Toho lze snadno dosáhnout například použitím editoru *WinEdt*. Druhou možností je použití vývojového prostředí *Eclipse*, které je také konfigurovatelné a existuje pro něj TinyOS plugin. Ten lze snadno dohledat na internetu, kde je také podrobně popsáno, jak ho do prostředí doinstalovat. Praktické zkušenosti s tímto nástrojem uvádím v příloze.

4.2 Vlastní návrh

Na základě znalostí získaných o TinyOS a programování pomocí jazyka *nesC* jsem vytvořil aplikaci, která je schopna získávat z navrženého hardwaru *GPS/ACC Board* data o aktuální pozici a posílat tyto informace bezdrátově přes radio modul *ZigBee*. Ten je implementován na přípravku Tmote Sky. Pro demonstraci získaných dat je pak jedno zařízení připojeno k PC a zasílá data přes seriovou linku rozhraní USB. Pro tento účel jsem také zhotoval aplikaci v jazyce C#, která má za úkol přijímaná data zobrazit. Blokové schéma přenosu dat je znázorněno na obrázku 4.4.



Obrázek 4.4: Přenos dat do PC

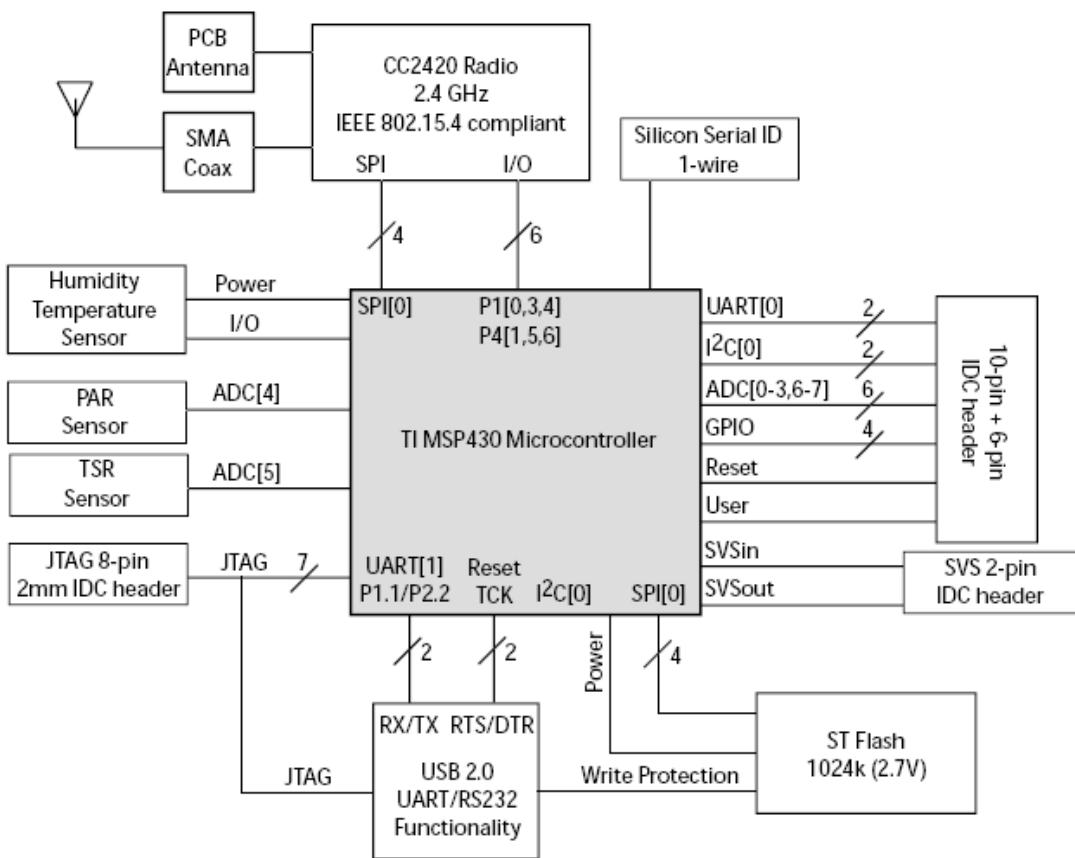
4.2.1 Aplikace pro Tmote Sky

Aplikace má za úkol cyklicky získávat data z GPS modulu a z akcelerometru a následně je posílat bezdrátově všem ostatním zařízením v dosahu. Program lze tedy rozdělit na několik částí a to na:

- seriová komunikace pro GPS data

- digitální vstupní data z akcelerometru
- Radiová komunikace

Tyto části zde podrobněji popíšu a naznačím hlavní strukturu celé aplikace. Pro lepší porozumění jednotlivým blokům uvádím na obrázku 4.5 blokové schéma modulu Tmote Sky s procesorem MSP430 od Texas Instruments. Pro náš účel je nejdůležitější rozšiřující konektor, kde jsou vyvedeny piny pro napojení přídavného hardwaru a radiový modul používaný pro bezdrátovou komunikaci. Pro programování modulů a pro přenos dat do PC jsem používal USB konektor, který je napojen na rozhraní UART1 mikroprocesoru.



Obrázek 4.5: Blokové schéma modulu Tmote Sky

4.2.1.1 Struktura programu

Pro získávání dat z akcelerometru a z GPS jsem sestavil dva samostatné moduly. To z toho důvodu, abych zpřehlednil celou strukturu a zjednodušil programování. Oba programy

mají shodnou strukturu a proto popíší pouze jeden z nich. Aplikace pro GPS se skládá ze tří souborů. Konfigurační soubor *TestUart0.nc*, implementační modul *TestUart0P.nc* a hlavičkový soubor *IntMsg.h*. Konfigurační soubor zajišťuje komunikaci aplikace s dalšími moduly, implementační modul vykonává vlastní program a v hlavičkovém souboru jsou definovány některé proměnné a struktury. V celém programu je celkem použito devět modulů, některé z nich jsou generické komponenty a mohou mít více instancí. Celkem je tedy využito dvanáct rozhraní.

Nejdůležitějším rozhraním, které musí poskytovat každá aplikace je *StdControl*. Toto rozhraní poskytuje tři funkce a to *StdControl.init()*, *StdControl.start()*, *StdControl.stop()*. Je to z toho důvodu, že jako první se v každé aplikaci TinyOS spouští modul *Main*, který používá právě rozhraní *StdControl* a volá jeho tři funkce. Je to hlavní tělo programu a odtud může programátor spustit svou aplikaci. Každá z těchto funkcí je provedena pouze jednou a po vykonání program skončí. Utvoření hlavní smyčky, která bude cyklicky provádět kód je tedy na programátorovi. TinyOS, a jeho aplikace, je systém řízený událostmi a proto se k vytvoření hlavní smyčky nejčastěji používá periodického časovače, který po načítání zvolené hodnoty vyvolá událost, vynuluje se a začne čítat znovu. Ve vzniklé události se pak vykonává kód. Tak tomu je i v této aplikaci.

V inicializační části dojde k inicializaci proměnných a explicitně k inicializaci některých komponent, které překladač nerozpozná a nenastaví sám. Příkaz *StdControl.init()* je spíše přežitek z TinyOS 1.0, kde se musela inicializovat většina komponent a pro přehlednost se toto dělo v samostatné sekci. Ve verzi TinyOS 2.0 a Boomerang 2.0.4 již toto systém dělá za nás. Příkladem může být rozhraní *Leds*, u kterého se musel nejdříve zavolat inicializační příkaz a potom explicitně zhasnout všechny LED diody.

Po zavolení příkazu *StdControl.start()* spustím dva hlavní časovače, jeden s periodou 2000 ms, který má za úkol získávat data z GPS modulu a druhý s periodou 50 ms, který čte data z akcelerometru a hlídá případné velké hodnoty zrychlení (to se děje především kvůli detekci nárazu).

Příkaz *StdControl.stop()* se volá jako poslední příkaz celého programu a může sloužit například pro zastavení časovačů, vypnutí hardwarových přerušení atd.

V konfiguračním souboru je tedy v sekci *components* uvedena komponenta *Main*, která se váže na rozhraní *StdControl* poskytované modulem *TestUart0P*.

```
Main.StdControl -> TestUart0P;
```

V implementačním modulu pak k tomuto účelu musí být v sekci *provides* uvedeno rozhraní *StdControl*, které ukazuje na fakt, že modul *TestUart0P* toto rozhraní implemen-

tuje. Je to také jediné rozhraní, které je v této aplikaci poskytováno a zajišťuje spuštění programu.

```
module TestUart0P {  
    provides {  
        interface StdControl;  
    }  
}
```

4.2.1.2 Stručný úvod do GPS

Data z modulu GPS jsou posílána ve standardním formátu. Pro dobré porozumění o výměně dat po seriové lince bych se nejprve v této podkapitole zmínil okrajově o GPS komunikačních protokolech. Další informace o GPS lze získat například v (Wikipedia, 2007) nebo pak podrobněji v (Daid, n.d.).

GPS modul od firmy *Falcom* (viz kapitola Hardware) podporuje dva komunikační protokoly. *NMEA protocol* a *SiRF Binary Protocol*. Oba dva jsou uznávanými standardy, přičemž jako první vznikl *NMEA (National Marine Electronics Association) protocol*. Mezi oběma protokoly lze libovolně přepínat za běhu programu zasláním příslušné konfigurační zprávy. Ve své aplikaci používám protokol *NMEA* a proto se protokolem *SiRF* nebudu dále zabývat. V (SiRF, 2005a) a (SiRF, 2005b) lze nalézt podrobné informace a další odkazy k těmto protokolům a také strukturu příchozích a odchozích zpráv.

*\$GPGLL, 161229.487, 3723.2475, N, 12158.3416, W, 1, 07, 1.0, 9.0, M, , , 0000 * 18*

Name	Example	Units	Description
Message ID	\$GPGGA		GGA protocol header
UTC Time	161229.487		hhmmss.sss
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W Indicator	W		E=east or W=west
Position Fix Indicator	1		See Table 1-4
Satellites Used	07		Range 0 to 12
HDOP	1.0		Horizontal Dilution of Precision
MSL Altitude	9.0	meters	
Units	M	meters	
Geoid Separation		meters	
Units	M	meters	
Age of Diff. Corr.		second	Null fields when DGPS is not used
Diff. Ref. Station ID	0000		
Checksum	*18		
<CR> <LF>			End of message termination

Obrázek 4.6: Zpráva NMEA

Na obrázku 4.6 je vidět příklad zprávy ve formátu *NMEA* a v tabulce pak najdeme význam jednotlivých prvků. Je to výstupní zpráva z GPS modulu, která určuje geografickou polohu a některé další údaje. Každá zpráva je uvozena znakem „\$“, což značí začátek zprávy a obsah zprávy končí znakem „*“. Za tímto znakem pak následuje dvoubyтовý kontrolní součet a celá zpráva je zakončena dvojicí znaků *CR* (*Carriage Return*) a *LF* (*Line feed*). Tyto dva znaky musí nutně ukončovat i každou příchozí zprávu, jinak nebude akceptována.

Po hlavičce zprávy, která uvozuje, že jde o data s geografickou polohou následuje čas ve formátu UTC, souřadnice polohy, příznakový Byte určující mód měrených dat, počet využívaných satelitů pro měření, dále pak přesnost měření a informace o jednotlivých satelitech. Jednotlivé informace se oddělují čárkou.

Vstupní zprávy mají stejnou strukturu jako zprávy výstupní, pouze hlavička zprávy je lehce odlišná. Je uvozena klíčovým slovem \$PSRF a za ním následuje identifikační číslo zprávy, které označuje o jaký typ se jedná a podle toho modul vyhledává ve zprávě příslušná data. Pokud je zpráva zadána ve špatném tvaru, modul ohlásí chybná vstupní data a zpráva je ignorována. Dál pak pokračuje ve vysílání se stejným nastavením. Modul *Falcom JP-13* podporuje ve své verzi firmware sedum typů výstupních zpráv a sedum

typů vstupních zpráv.

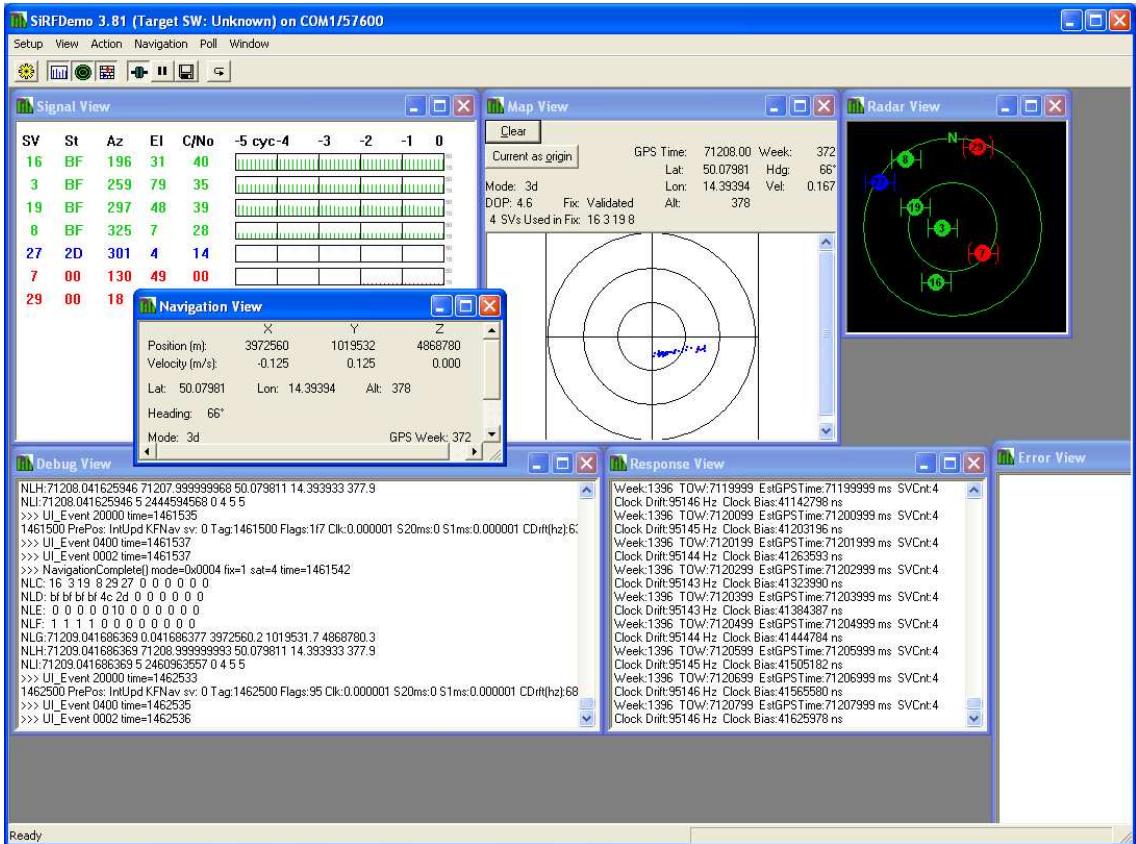
Option	Description
GGA	Time, position and fix type data.
GLL	Latitude, longitude, UTC time of position fix and status.
GSA	GPS receiver operating mode, satellites used in the position solution, and DOP values.
GSV	The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and SNR values.
MSS	Signal-to-noise ratio, signal strength, frequency, and bit rate from a radio-beacon receiver.
RMC	Time, date, position, course and speed data.
VTG	Course and speed information relative to the ground.

Obrázek 4.7: Výstupní zprávy protokolu NMEA

Message	MID ¹	Description
SetSerialPort	100	Set PORT A parameters and protocol
NavigationInitialization	101	Parameters required for start using X/Y/Z ²
SetDGPSPort	102	Set PORT B parameters for DGPS input
Query/Rate Control	103	Query standard NMEA message and/or set output rate
LLANavigationInitialization	104	Parameters required for start using Lat/Lon/Alt ³
Development Data On/Off	105	Development Data messages On/Off
Select Datum	106	Selection of datum to be used for coordinate transformations.
MSK Receiver Interface	MSK	Command message to a MSK radio-beacon receiver.

Obrázek 4.8: Vstupní zprávy protokolu NMEA

Společnost *Falcom* uvádí na svých stránkách volně k dispozici demonstrační software *SiRFDemo*, který umožňuje přes seriový port zpracovávat získaná data ze svých GPS modulů. Tohoto nástroje jsem hojně využíval při odlad'ování vlastní aplikace a testování vyrobené DPS osazené modulem *Falcom JP-13*. Bylo však nejprve třeba zhodnotit převodník napěťových úrovní. Na obrázku 4.9 je zobrazeno okno této aplikace, kde je vidět aktuální poloha, rychlosť, datum a čas, informace o satelitech a zprávy protokolu NMEA a SiRF.

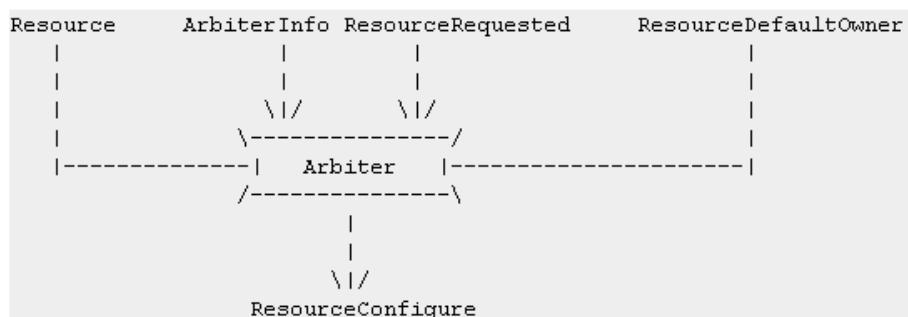


Obrázek 4.9: Softwarový nástroj SiRFDemo

4.2.1.3 Komunikace s GPS modulem

Jak již jsem zmínil po seriové lince se přenáší GPS data z GPS/ACC modulu. Jelikož se jedná o pevně definovaný formát zpráv, musel jsem v programu použít rozhraní nižší vrstvy, kde jsem přenášel postupně jednotlivé Byty. Po odvysílání nebo přijmutí každého Bytu je rozhraním indikována událost o dokončení a následně může být zpracován další Byte. Implementace vyšších vrstev seriové komunikace nebyla možná, protože ty přidávají do zprávy vlastní „hlavičku“ a tu zároveň i vyžadují při zpracování.

Druhým úskalím při seriové komunikaci je fakt, že na modulu Tmote Sky seriová linka sdílí stejnou datovou sběrnici s ADC modulem, SPI rozhraním a Radio modulem. K tomuto účelu jsou ale v TinyOS implementována rozhraní *Resource*, *ResourceCmd* a *ResourceCmdAsync*, které slouží na „rezervaci“ společných hardwarových prostředků. Všechny tři jsou velice podobné a liší se ve funkcích, které poskytují.



Obrázek 4.10: Arbitráž sdílených hardwarových prostředků

Na obrázku 4.10 je vidět zjednodušeně princip přidělování společných prostředků. Všechny tyto prostředky mají svého *softwarového arbitra*, který je konfiguruje a předává informace nadřazeným vrstvám přes rozhraní *Resource*. Pokud tedy přijde požadavek ze strany aplikace pro rezervaci daného zdroje, arbitr tento zdroj obsadí a předá nadřazené vrstvě informaci o přidělení zdroje. Ta jej pak může využívat a následně musí dát pokyn arbitrovi k jeho opětovnému uvolnění. Pokud při požadavku je již zdroj obsazen, záleží na prioritě požadavku a podle toho bud' arbitr přidělí daný prostředek novému vlastníkovi či nikoliv. Samozřejmě pomocí událostí informuje příslušné nadřazené vrstvy. Podrobnosti o přidělování prostředků lze nalézt v Online dokumentaci, viz (*TEP files*, 2007) TEP108.

Ve své aplikaci používám arbitra s rozhraním *ResourceCmd*, které nalezneme v modulu *MSP430ResourceUART0C*. Tento modul je generická komponenta a proto lze vytvořit instance téhoto rozhraní a využívat je nezávisle na sobě. Pro seriovou komunikaci využívám dvě rozhraní arbitra a to jedno pro příjem dat a jedno pro vysílání dat. V konfiguračním souboru jsou pak navázány na příslušný modul.

```

components
  new MSP430ResourceUART0C() as ResourceCmd,
  new MSP430ResourceUART0C() as ResourceCmd1,
  :
  TestUart0P.ResourceSend -> ResourceCmd;
  TestUart0P.ResourceReceive -> ResourceCmd1;

```

Jak je vidět z tohoto příkladu, samotnou aplikaci využívající tato rozhraní najdeme v modulu *TestUart0P* a to pod názvem *ResourceSend* a *ResourceReceive*. Pro testování a ladění jsem ještě používal stejný typ rozhraní pod názvem *ResourceDebug*.

V implementačním modulu tedy najdeme ve výčtu použitých rozhraní i tato dvě, musí být ale uvedeno od kterého typu jsou odvozena. Ten si pak překladač sám doplní v konfiguračním souboru pro správné navázání.

```
uses {
    interface ResourceCmd as ResourceSend;
    interface ResourceCmd as ResourceReceive;

    :
}

}
```

Dále pak z ostatních komponent využívám k seriové komunikaci rozhraní *UartControl* a *UartFeedback* modulu *HPLUSART0M*, které slouží k nastavení seriové linky a k posílání a přijímání jednotlivých Bytů včetně ovládání přerušení.

V příkazu *StdControl.start()* nejprve provedu nastavení GPS modulu zasláním několika NMEA zpráv. Jde především o nastavení linky (přenosová rychlosť, parita, počet stop-bitů) a o nastavení typu zasílaných zpráv. GPS modul po resetování zasílá každou sekundu pět typů zpráv, což je ale zbytečné. Pro náš účel jsou dostačující pouze informace o poloze, GPS datu a čase a indikátor o platnosti přenášených dat. Ostatní zprávy pak vyřadíme zasláním konfiguračních zpráv. K tomuto účelu jsem vytvořil funkci *sendSerial*, které jako parametr předávám pole Bytů obsahující příslušnou zprávu a počet prvků pole. Ta má potom ve svém těle cyklus, kde čeká na odeslání předchozích zpráv. Pokud není odesílána žádná zpráva, je nastaven příznak, který zablokuje poslání další zprávy a je uvolněn až na konci této procedury. V dalším kroku alokuji dynamické pole *sBuff*, do kterého nakopíruji zprávu předávanou jako parametr *sendSerial* a následně spočtu kontrolní součet funkcí *checkSum*. Kontrolní součet se spočítá jako bitový XOR a následně je rozdělen na dva Byty podle standardu NMEA. Tento součet pak spolu s ukončovacími znaky přidám do pole *sBuff* a nakonec pošlu žádost arbitrovi s rozhraním *ResourceSend* o rezervaci hardwarové sběrnice.

Poté co arbitr rezervuje sběrnici, ohlásí obsazení událostí *ResourceSend.granted*. Od tohoto okamžiku může aplikace posílat data po seriové lince. Nejdříve je však nutno provést nastavení seriové linky pomocí rozhraní *UartControl*.

```
//-- Make any necessary UART changes
call UartControl.disableSPI();
call UartControl.disableI2C();
```

```

call UartControl.enableUARTTx();
call UartControl.enableTxIntr();
call UartControl.setClockSource(SSEL_SMCLK);
call UartControl.setClockRate(UBR_SMCLK_38400, UMCTL_SMCLK_38400);

```

Toto rozhraní zároveň ovládá komunikaci po SPI a I2C. Nejdříve tedy vyřadíme tyto dva typy komunikace a povolíme pouze vysílací část seriové linky. Zároveň povolíme přerušení, které nám po odvysílání každého Bytu vyvolá událost v programu přes rozhraní *UartFeedback*.

Po konfiguraci seriové linky odešleme první Byte z proměnné *sBuff*. Každý další Byte pak odešleme až z události *UartFeedback.txDone()*, která se vyskytne po odvysílání Bytu. Po poslání celé zprávy je ještě potřeba uvolnit sběrnici zavoláním příkazu *ResourceSend.release()*.

Po nastavení GPS modulu již není dále potřeba zasílat zprávy a pouze tedy přijímáme data. To se děje cyklicky s periodou 2000 ms po přidělení linky přes rozhraní *ResourceReceive* v každém cyklu. Nejprve je opět provedena konfigurace seriové linky. Je povolena pouze přijímací část a příslušné přerušení.

V první fázi systém přijímá Byty a čeká na úvodní znak zprávy „\$“. Po příchodu tohoto znaku se v události *rxDone()* rozhraní *UartFeedback* ukládají další příchozí znaky přímo do struktury, která se používá pro rádiovou komunikaci. Mezitím musí být utvořena čekací smyčka, která zajišťuje stále blokování příslušné sběrnice. Ve smyčce se čeká na ukončovací znak zprávy „*“ po kterém následují ještě dva Byty kontrolního součtu. Po přijetí celé zprávy se ukončí čekací smyčka a arbitr uvolní daný HW zdroj. V případě, že nepřijmeme znak „*“ se smyčka ukončí po naplnění pole o velikosti uvedené v konstantě *RECEIVED_BYTES*. Přijatá zpráva je pak bezdrátově distribuována do ostatních Tmote Sky modulů, v našem případě pak z druhého modulu do PC.

4.2.1.4 Komunikace s akcelerometrem

Tmote Sky moduly mají na svých rozšiřujících konektorech vyvedeny čtyři digitální vstupy/výstupy, které může programátor libovolně použít. Z důvodu ušetření místa jsou ale piny sdíleny i s dalšími hardwarovými prostředky a proto je třeba dát pozor na jejich konfiguraci. Dva z nich jsou vyvedeny na 10-ti pinovém konektoru - GPIO0 a GPIO1, které jsou sdíleny s analogovými vstupy AD převodníku. Přes tento konektor je připojen náš GPS/ACC modul. Druhé dva digitální vstupy/výstupy (GPIO2, GPIO3) se nachází na 6-ti pinovém rozšiřujícím konektoru, který je ponechán volný. Aby piny GPIO0 a

GPIO1 byly použity jako digitální vstup/výstup musí být na desce Tmote Sky propojeno místo R14 pro GPIO0 a R16 pro GPIO1. Protože se jedná o technologii SMT jsou piny velice malé a je třeba dát velký pozor na bezchybné propojení. Bohužel to se v méém případě nepovedlo a na těchto propojkách vznikal příliš velký odpor a následkem toho značný úbytek napětí. Funkce těchto dvou digitálních vstupů pak nebyla úplně v pořádku a proto jsem musel použít vstupy GPIO2 a GPIO3 an 6-ti pinovém konektoru, které jsou standardně hardwarově nastaveny jako digitální vstupy/výstupy.

Ve své aplikaci používám pro čtení PWM signálu z akcelerometru dva piny. Akcelerometr má dva výstupy a měří zrychlení ve směru X a Y. Pro ovládání veškerých digitálních vstupů/výstupů slouží v TinyOS rozhraní *MSP430GeneralIO* a *MSP430Interrupt*. Ty jsou implementovány v následujících dvou modulech a navázány na příslušný port.

components

```
MSP430InterruptC,
MSP430GeneralIOC,
```

:

```
GPIO0P.Port23 -> MSP430GeneralIOC.Port23;
GPIO0P.Inte23 -> MSP430InterruptC.Port23;
GPIO0P.Port26 -> MSP430GeneralIOC.Port26;
GPIO0P.Inte26 -> MSP430InterruptC.Port26;
```

Jak je vidět v modulu *GPIO0P* jsem si tato rozhraní pojmenoval a navázal na *Port23* a *Port26* příslušné komponenty. Je to z toho důvodu, že rozhraní *MSP430GeneralIO* má vytvořeno několik instancí, kde každá zastupuje jeden digitální vstup/výstup mikroprocesoru MSP430 a jsou pojmenovány podle příslušného portu a pinu (Port20 je například port 2 a pin 0 tohoto portu mikroprocesoru). Stejně je tomu u rozhraní *MSP430Interrupt*. Definici těchto instancí lze nalézt v hlavičkovém souboru:

/opt/tinyos-1.x/tos/platform/msp430/MSP430GeneralIO.h v instalaci Cygwinu. Ve schématu Tmote Sky, viz datasheet (Mote, 2006), je nutno dohledat, které piny reprezentují vyvedené GPIO.

V aplikaci je tedy nejdříve provedeno softwarové nastavení pinů - zda se jedná o standardní I/O pin nebo mu přiřazená nějaká specifická funkce mikroprocesoru, dále se pak určí, že se jedná o vstupní pin nikoliv o výstupní a v poslední řadě se povolí přerušení od tohoto vstupu. Kód pro ovládání obou použitých vstupů, na které přichází

měření z akcelerometru ve dvou osách je shodný a proto popíšu ovládání pouze jednoho vstupu.

Po inicializaci je v hlavní proceduře spuštěn časovač, který má periodu 50 ms. V těle události vzniklé po načítání do zadané hodnoty je pak část kódu, která ovládá čtení dat z akcelerometru. Nejprve se v každém cyklu rozhoduje, zda-li je právě povoleno přerušení od vstupu a podle toho se vykonává jistá část kódu. Přerušení se povoluje periodicky v tomto cyklu s periodou 50 ms. To je dostatečně krátká doba na to, aby se zachytily i velký okamžitý nárust zrychlení. Není nezbytně nutné zachytit maximální špičku zrychlení při nárazu, která nám i při této periodě může uniknout. V takto krátkém časovém intervalu ale jistě zjistíme dostatečně velké zrychlení na to, abychom tuto událost vyhodnotili jako náraz či případné prudké dobrždění.

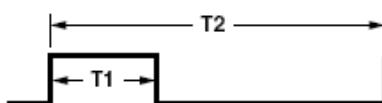
Při povolení přerušení tedy procesor vyvolá událost *Inte26.fired()*. Ta je vyvolána nástupní hranou příchozího impulsu. K tomu, abychom zachytily sestupnou hranu musíme přenastavit systém přerušení a povolit ho na opačnou hranu. K tomu slouží tato sekvence kódu:

```

call Inte23.disable();
call Inte23.clear();
//----- edge flipping
if (edgeX == TRUE)
    edgeX = FALSE;
else
    edgeX = TRUE;
call Inte23.edge(edgeX);
call Inte23.enable();

```

Nejdříve vymažeme aktuální příznak přerušení pro daný port, zjistíme, jestli bylo vyvoláno na vzestupnou nebo sestupnou hranu impulsu a povolíme znovu přerušení s opačnou hranou. Tento mechanismus nám tedy zachytává důležité časové okamžiky PWM signálu, které můžeme změřit pomocí časovače.



Obrázek 4.11: Výstupní PWM signál akcelerometru

Hodnota aktuálního zrychlení se vypočítá dle obrázku 4.11 jako:

$$g = (T1/T2 - 0,5)/4[\%]$$

podrobněji v datasheetu (Analog, 2002), přičemž Hodnota T2 je závislá na odporu R_{set} , který figuruje v zapojení. V mém případě je perioda T2 přibližně 1 ms. Dle datasheetu mikroprocesoru MSP430 (TI, 2006) nalezneme, že hodiny mikroprocesoru jsou taktovány v řádu desítek ns, tudíž je zde dostatečná rezerva pro takovouto periodu vstupního signálu.

Při každém přerušení na nástupnou hranu vstupu je zapnut časovač. Při setupné hraně impulsu se přečte aktuální načítaná hodnota a uloží se. Časovač je vypnuto. Takto se po celou dobu čtení dat z akcelerometru integruje doba aktivní úrovně signálu neboli T1. Doba T2 je pak 50 ms a z téhoto hodnot můžeme vypočítat průměrnou hodnotu zrychlení v tomto časovém intervalu. Samozřejmě je měření zatíženo určitou chybou, která se zvětšuje použitou metodou měření. Pro naše účely, kdy potřebujeme detektovat pouze velké hodnoty zrychlení, můžeme ale prohlásit získaná data za dostatečně přesná.

Jednotlivé hodnoty zrychlení jsou ukládány do pole. V případě, že zrychlení dosáhne zvolené prahové úrovně jsou data ihned poslána přes bezdrátový modul. Při běžných hodnotách zrychlení se posílají společně s daty GPS.

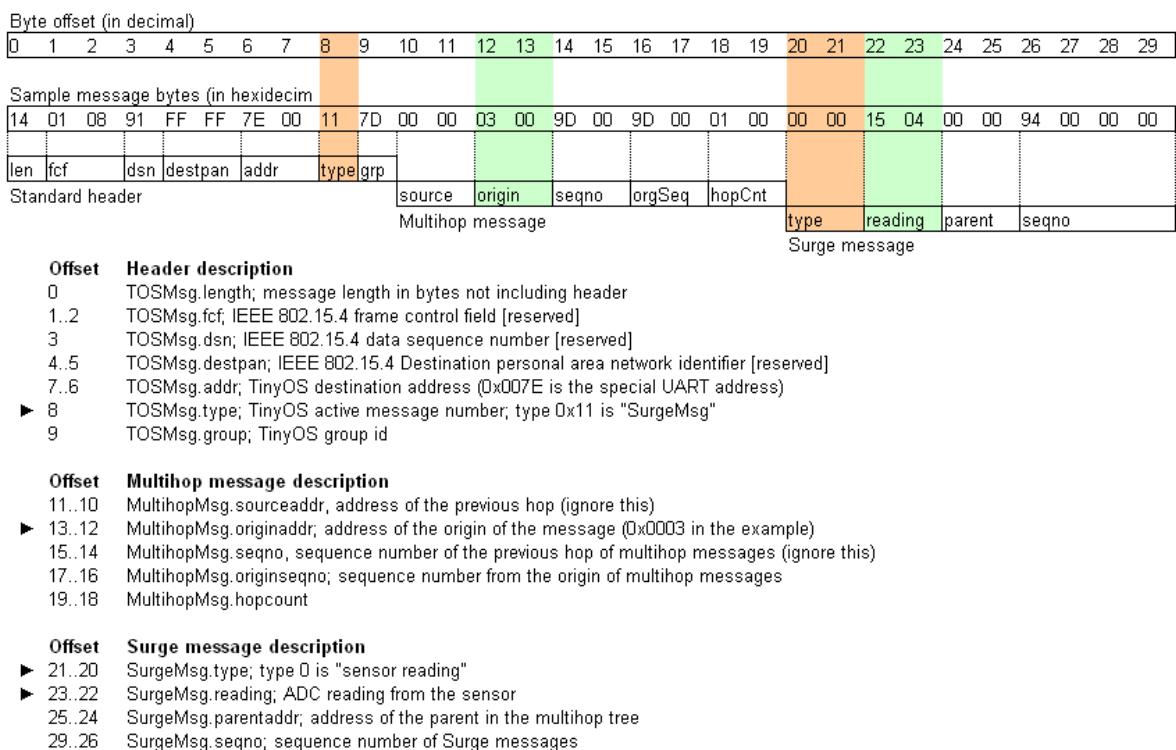
4.2.1.5 Bezdrátová komunikace mezi moduly Tmote Sky

Na obrázku 4.5 je vidět připojení radiového modulu CC2420 od firmy Chipcon. Pro naše účely byla plně dostatečná integrovaná anténa, v praxi by pak kvůli umístění modulu a kvůli robustnosti systému bylo vhodnější instalovat SMA konektor a připojit externí anténu. Tento modul je připojen k mikroprocesoru přes rozhraní SPI a několik digitálních vstupů/výstupů. Má softwarově nastavitelný vyzařovací výkon, zároveň může být procesorem vyřazen z provozu pro úsporu energie.

Rozhraním pro posílání zpráv přes ZigBee modul, které využívám ve své aplikaci je *SendMsg[AM_INTMSG]*. Je implementováno v komponentě *GenericCom* nebo *SPC*. První uvedená je komponenta pro starší aplikace fungující pod TinyOS 1.0 a ve verzi Boomerang 2.0.4 byla zachována pro kompatibilitu, nicméně její konfigurační soubor se váže na implementaci *SPC*. V obou případech zde najdeme také rozhraní pro přijímání zpráv. Dále je vidět, že rozhraní *SendMsg* je parametrizované s parametrem *AM_INTMSG*. Tento parametr si můžeme představit jako port při komunikaci mezi PC. Odesílatel specifikuje AM číslo při odesílání zprávy. Na straně přijímače je pak vyvolána událost o přijetí

zprávy v aplikacích, které mají registrované stejné AM číslo. V jedné aplikaci pak může být definováno i více druhů zpráv lišících se AM číslem. Musí pak být samozřejmě použit příslušný počet rozhraní. To umožňuje například ovládání dvou typů zpráv od různých zařízení.

Zprávy, které se přenáší přes tato rozhraní se nazývají *TOS_Msg*. Je to definovaná struktura s vlastní hlavičkou a polem datových Bytů.



Obrázek 4.12: Formát zpráv TOS_Msg

V tabulce na obrázku 4.12 je vidět pořadí a význam jednotlivých Bytů. Prvních deset Bytů je hlavička standardní zprávy, za kterou následuje pole dat. Tuto strukturu používám ve své aplikaci. Další Byty hlavičky jsou přítomny pouze při definování speciálního typu zprávy.

Důležitým souborem pro přenastavení počtu přenášených Bytů je hlavičkový soubor *AM.h*. Můžeme ho najít ve struktuře Cygwinu v adresáři:

/opt/moteiv/tos/lib/CC2420Radio.

Jako standardní délka datové zprávy je přednastavena hodnota 28 Bytů. Maximální možná délka datového balíku je však 116 Bytů (Chipcon uvádí u svého modulu maximální

počet 128 Bytů - 10 Bytů TOS hlavička - 2 Byty kontrolní součet). Je třeba si však uvědomit, že při posílání velkého objemu dat se snižuje propustnost bezdrátové sítě.

V případě potřeby lze dále měnit v určitém rozmezí výstupní vysílací výkon rádiového modulu a vysílací frekvenci. To lze provést pomocí rozhraní *CC2420Control*, které najdeme v modulu *CC2420ControlM*, zavoláním funkcí

```
command result_t TunePreset( uint8_t rh, uint8_t channel );\\
command result_t SetRFPower( uint8_t rh, uint8_t power );.
```

Výstupní výkon lze měnit v rozsahu hodnot 1 až 31, přičemž 1 představuje výkon -25 dBm a 31 maximální výkon 0 dBm. Pro výstupní frekvenci lze pak použít jeden z kanálů od 11 do 26, přičemž výstupní frekvence se vypočítá jako

$$Freq = 2405 + 5(k - 11) MHz \text{ pro } k = 11 \dots 26$$

Pro demonstraci aplikace jsem nekladl žádné nároky na distribuci zpráv různým uživatelům a zprávy jsem posílal jako *broadcast* s adresou 0xffff. Pro reálnou situaci se ale dá dobře využít skutečnosti, že každý modul má svou specifickou rádiovou adresu a lze zasílat zprávy jen na vybrané moduly. Tím se také samozřejmě sníží propustnost sítě.

Jak již jsem zmínil dříve, GPS data jsou ukládána do struktury rádio zprávy *TOS_Msg.data[index]* při přijímání jednotlivých Bytů. Po uložení všech Bytů GPS dat a uvolnění hardwarové sběrnice je zavolána funkce *sendMessage()*, která za tato data nakopíruje naměřená data z akcelerometru. Celý tento balík i s hlavičkou je pak poslan příkazem společně s indikací.

```
if (call moje.send(TOS_BCAST_ADDR, index, &m_tosmsg) == SUCCESS) {
    call Leds.greenOn();
}
```

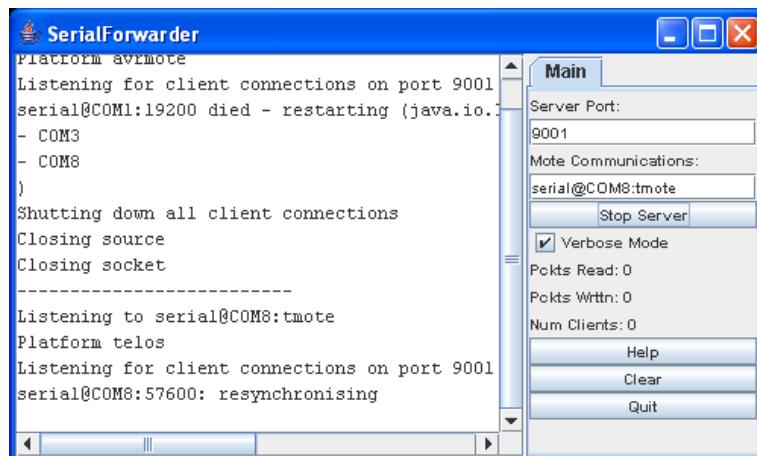
Jako parametry této funkce se zadává seznam adres příjemců, počet datových Bytů přenášené zprávy a adresa ukazatele na zprávu *TOS_Msg*. Po odeslání všech Bytů zprávy aplikace obdrží událost s úspěšném odeslání, kde se opět provede indikace.

```
event result_t moje.sendDone(TOS_MsgPtr msg, result_t success) {
    call Leds.greenOff();
    return SUCCESS;
}
```

4.2.2 Aplikace pro PC

Pro zobrazení přenášených dat jsem zhotovil aplikaci pro PC, která přijímá data z Tmote modulu připojeného k USB portu. Cyklicky pak zobrazuje naměřené hodnoty z GPS modulu. Aplikace je napsána pro prostředí Microsoft Visual .NET 2.0 v jazyce C# a tudíž je určena pro operační systém Windows. K tomu, aby bylo možno přijímat data, musí být v připojeném Tmote modulu aplikace, která bude posílání dat do PC obstarávat. Je to však standardní nástroj systému TinyOS. Aplikace se jmenuje *TOSBase* a najdeme ji v adresáři /opt/moteiv/apps/TOSBase. Pozor, v instalaci Boomerang 2.0.4 najdeme tuto aplikaci ještě ve struktuře /opt/tinyos-1.x/apps/TOSBase, ta je ale určena pro standardní instalaci TinyOS 1.2 a nebude s modulem Tmote fungovat. Stejně tak tomu může být i v případě dalších nástrojů a vždy je lepší použít verzi v adresáři /moteiv než z adresáře //tinyos-1.x pokud to jde. Většina aplikací je však funkčních.

Dalším standardním nástrojem v systému TinyOS, který využívám ve své práci je *SerialForwarder*. Je to aplikace napsaná v Javě, která umí číst vysílaná data Tmote modulem ze seriového USB portu a vytváří na PC lokální server s definovaným komunikačním portem, ze kterého lze číst data.



Obrázek 4.13: Aplikace SerialForwarder

Aplikace se nachází ve struktuře TinyOS spolu s ostatními Java nástroji v podadresáři /tools/java. Všechny Java komponenty jsou přeloženy při instalaci TinyOS, v případě nefunkčnosti lze tyto nástroje znova přeložit pomocí linuxového příkazu *make*. *SerialForwarder* se spouští v prostředí Cygwin příkazem

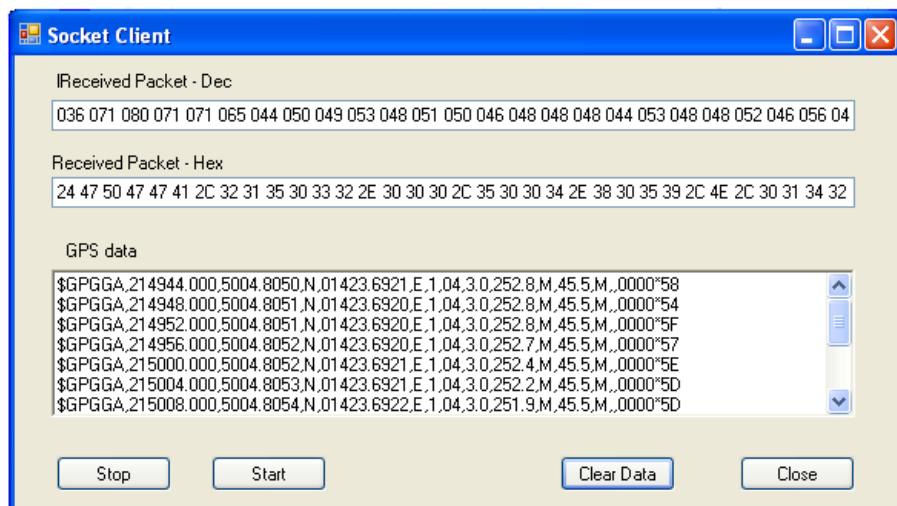
```
java net/tinyos.sf.SerialForwarder
```

Jako parametry při spuštění lze zadat číslo seriového portu a přenosovou rychlosť, jednodušší je však nastavit tyto hodnoty přímo v okně aplikace. Můžeme zde také nastavit číslo portu na kterém server čeká na příchozí spojení, tzn. na kterém portu je připojen Tmote Sky modul. Z lokálního serveru pak získáme údaje z bezdrátové sítě přes jednoduchý komunikační protokol SF.

Jedná se o TCP komunikaci pomocí socketů, jak je naznačeno na obrázku 4.4. Server poslouchá na daném portu, v případě, že se na tento port přihlásí libovolná aplikace, proběhne mezi oběma stranami úvodní komunikace, pak při příchozích datech z Tmote modulu server posílá data na socket aplikace.

Aplikaci jsem pojmenoval *SocketClient*. Obsahuje několik základních souborů a to *Program.cz*, kde se nachází hlavní procedura *Main* celé aplikace, dále pak *Form1.cs* obsahující třídu *Form1* zděděnou od základní třídy *Form*. Zde je pak napsána převážná část uživatelského kódu. Třetím souborem je *Form1Designer.cs*, v němž jsou vytvořeny instance všech objektů vytvořených v projektu a jsou jim přiřazeny názvy a specifické vlastnosti, jako je například u grafických objektů umístění, zobrazovaný text, obslužné události atd. Tato část kódu je převážně generovaná automaticky Microsoft Visual Studiem při nastavování parametrů programátorem v grafickém okně a do tohoto souboru nemusíme zasahovat příliš často.

V hlavní proceduře je pak inicializováno vizuální prostředí a spuštěno grafické rozhraní *Socket Client* celé aplikace.



Obrázek 4.14: Grafické okno aplikace pro PC

Pro připojení na server využívám objekt *TcpClient* nacházející se v knihovnách Sys-

tem.Net a System.Net.Sockets a dále je potřeba na čtení dat ze socketu objekt typu *NetworkStream*. Obě komponenty jsou vytvořeny v konstruktoru objektu *Form1* a klient se připojí na server **localhost**, **port 9001**. Do proměnné *NetworkStream* se pak uloží pointer na datový tok z tohoto socketu.

V první fázi server pošle inicializační Byte a klient je povinen odpovědět dvojicí Bytů.

```
+-----+
| SFP Cookie | SFP Version |
+-----+
```

Obrázek 4.15: Inicializace SF protokolu

SFP Cookie má vždy stejnou hodnotu 84 a určuje, že se jedná o SF protokol. Druhý Byte SFP Version slouží potom pro identifikaci hardwarové platformy ze které jsou přenášena data.

- 32: S touto verzí nedostaneme od serveru informace o HW modulu.
- 33: S touto verzí následuje packet s identifikací platformy.

Jelikož používáme pouze Tmote Sky moduly stačí použít verzi 32. Tyto dva Byty odesleme jako odpověď na server a následně můžeme přijímat požadovaná data.

Cyklické čtení je zajištěno opět komponentou časovače, která v intervalu 500 ms zjišťuje zda-li jsou k dispozici nějaká data. Pokud ne, čeká se další cyklus. V případě, že server poslal nějaká data, je znova vytvořeno pole, do kterého se poslané Byty uloží. Objekt *NetworkStream* nemá bohužel podporu pro zjištění celkového počtu Bytů v datovém toku, tudíž velikost pole nemůže být definována dynamicky. Je tedy vytvořen dostatečně velký buffer s pevným počtem Bytů. To ale není problém vzhledem k paměťovým prostředkům PC.

První Byte v každé posloupnosti udává celkový počet Bytů ve zprávě *TOS_Msg*. V případě, že nečteme data ze socketu pravidelně, může se ve frontě nahromadit více zpráv. Proto po přečtení dat zjišťuji počet zpráv jako

$$messageCount = bytes / data[0]$$

kde proměnná *bytes* představuje celkový počet přečtených Bytů a proměnná *data[0]* udává počet Bytů ve zprávě *TOS_Msg*.

Pro tento počet zpráv pak udělám vnější smyčku a zobrazím je všechny v jednom časovém cyklu Timeru. Dále vím, že hlavička obsahuje 10 Bytů, proto začínám parsovat

data od 11. Bytu a postupně při procházení zvyšuji index. Do horních dvou textových polí na obrázku 4.14 ukládám jednotlivé Byty v číselné a hexa podobě. Z těchto dat pak lze spočítat hodnotu aktuálního zrychlení dle vzorce pro výpočet zrychlení uvedeného v kapitole 4.2.1.4. Tato dvě pole lze také využít při debugování. Debugování mikroprocesorů je obecně nelehká záležitost a proto bez zvláštních prostředků se zde dají zobrazovat hodnoty proměnných.

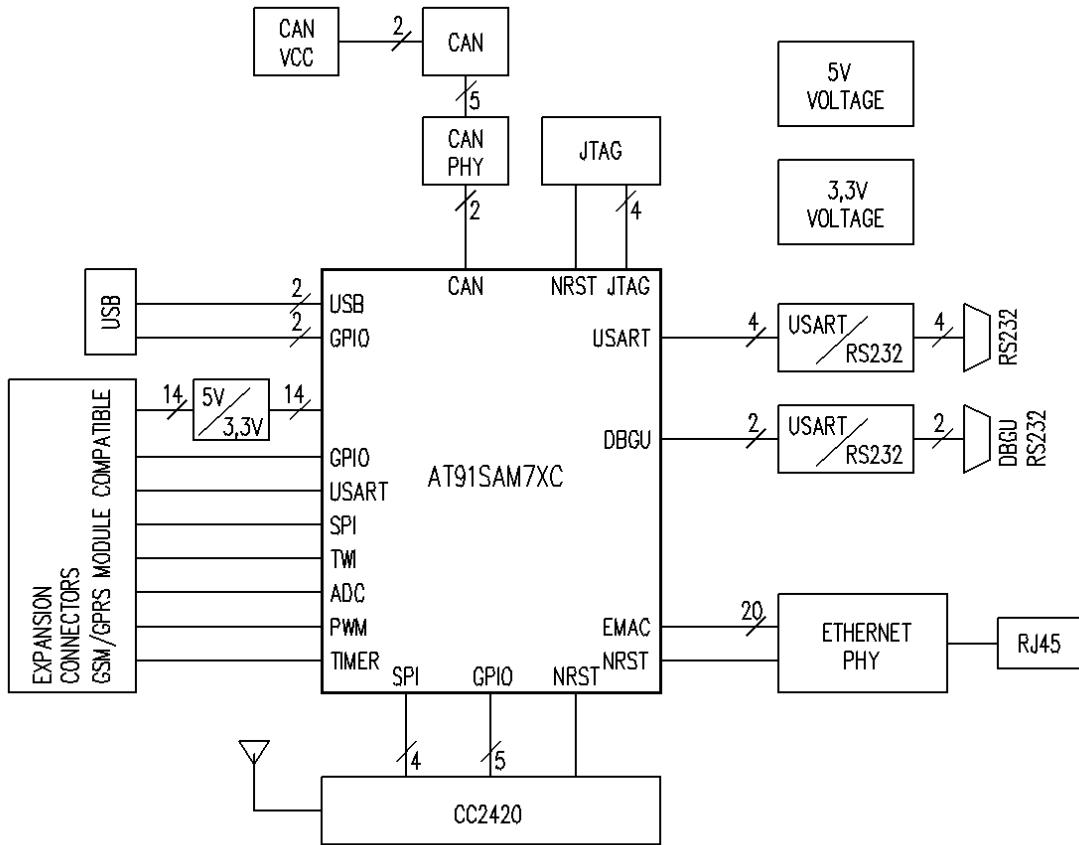
Další pole slouží pro zobrazování GPS dat, kde pro možnosti testování přenáším celou zprávu NMEA. Pro ovládání slouží čtyři tlačítka, které mají možnost přerušit cyklické čtení dat zastavením timeru a jeho opětovným zpuštěním, dále vyčištění obsahu textových polí a v poslední řadě ukončení aplikace.

Kapitola 5

Rozšiřující Hardware

5.1 Řídící jednotka

Protože se ukázalo, že výrobky Tmote Sky nejsou pro tento varovný systém dostačující (především chybí podpora sběrnice CAN), navrhli jsme nový hardwarový modul. Tento modul je navržen univerzálně a má sloužit pro testovací účely. Návrh vychází z diplomových prací (Musil, 2003), (Černý, 2005), které se zabývají návrhem řídící jednotky pro komunikaci s CAN sběrnicí a návrhem rozšiřujícího modulu GSM/GPRS. Protože základem této řídící jednotky je dnes již zastaralý typ procesoru od firmy Motorola, bylo potřeba nahradit jádro celé desky novým výkonným mikropočítáčem a navrhnout nové schéma. Základním požadavkem při návrhu však bylo zachovat kompatibilitu s rozšiřujícím modulem.



Obrázek 5.1: Blokové schéma zapojení

Na obrázku 5.1 je vidět blokové schéma navrhované jednotky. Jádrem je procesor AT91SAM7XC od firmy Atmel. Pro komunikaci se svým okolím používá několik typů rozhraní:

- RS232

Toto rozhraní je vyvedeno na standardním 9-ti pinovém konektoru pro DSUB, slouží pro seriovou komunikaci. Na desce je integrován převodník napěťových úrovní.

- Debugging Interface

Debugovací rozhraní je standardem u novějších procesorů a slouží k ladění programu mikroprocesoru. Lze ho připojit k seriové lince PC a proto je opět integrován napěťový přizpůsobovač.

- JTAG Interface

Slouží pro testování a programování mikroprocesoru.

- CAN

Komunikační rozhraní pro sběrnici CAN. Je to průmyslová sběrnice používaná například v automobilech. Pro připojení na sběrnici byl na desce integrován sběrnicový budič spolu s indikací stavů.

- USB

Seriové rozhraní, zde slouží pro připojení k zařízením typu USB master (například k PC).

- ZigBee

Pro bezdrátovou komunikaci je deska navržena s radiovým modulem ZigBee. Byl použit chip CC2420 od firmy Chipcon.

- Ethernet

Připojení k Ethernetové síti obstarává řadič EMAC mikroprocesoru spolu s fyzickým budičem signálů.

- Rozhraní pro rozšiřující GSM/GPRS modul

Zachovává kompatibilitu s hardwarovým GSM/GPRS modulem navrženým v diplomové práci (Černý, 2005). Z tohoto důvodu je rozšiřující konektor uzpůsoben připojení k tomuto hardwaru. Bylo potřeba přizpůsobit napěťové úrovně z 3,3V logiky (nová řídící jednotka) na 5V logiku mobilní jednotky.

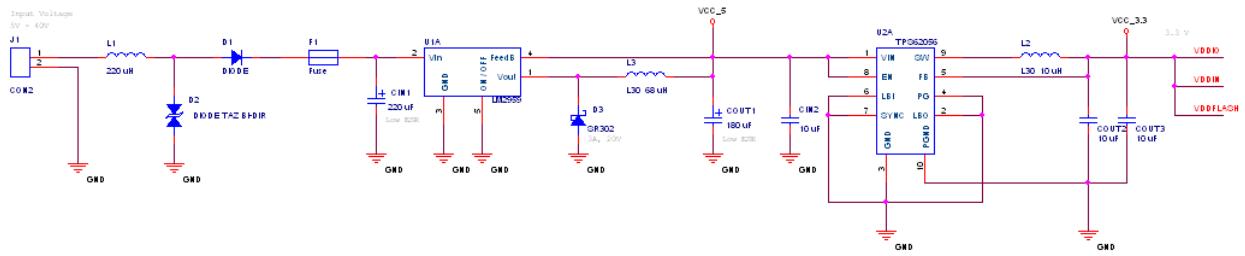
- Ostatní typy rozhraní

Dále jsou na rozšiřující konektory vyvedeny všechny ostatní rozhraní a funkční bloky procesoru - USART, SPI, TWI, GPIO, ADC, PWM, Timer a další.

Procesor má celkem 64 digitálních vstupů/výstupů, které jsou sdílené a jejich funkce se dá nastavit softwarově přes řídící registry. Aby bylo možno integrovat všechna uvedená rozhraní je potřeba tyto konfigurovatelné vstupně/výstupní piny sdílet a tudíž to vylučuje použití některých rozhraní současně. Příkladem může být Ethernetová komunikace (používá 20 pinů procesoru) a současně připojení rozšiřujícího GSM/GPRS modulu (použito 14 pinů procesoru). Některé jejich signálové piny jsou společné. V dalších podkapitolách popíšu jednotlivé funkční bloky návrhu a vysvětlím jejich provázanost.

5.1.1 Napájení

Schéma napájecí části obvodu je vidět na obrázku 5.2.

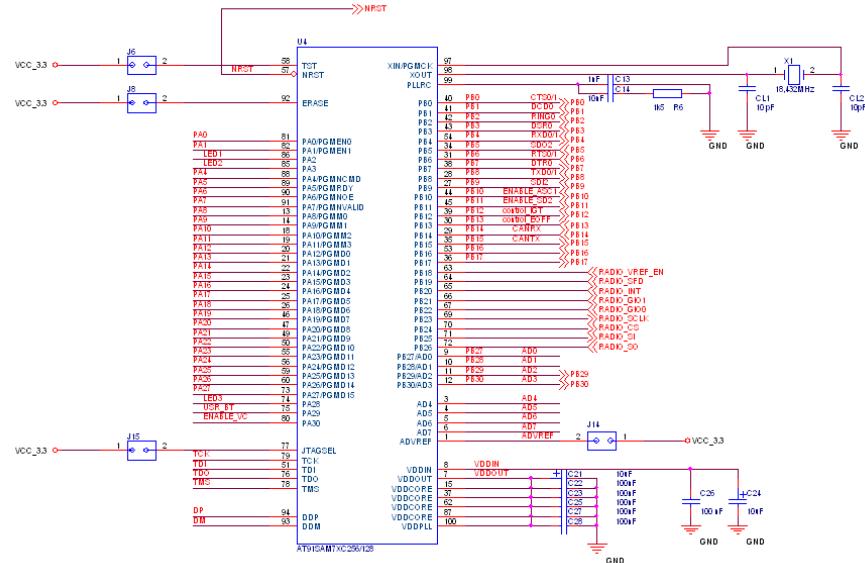


Obrázek 5.2: Napájecí obvod

Celá deska může být napájena ze stejnosměrného zdroje napětí v rozmezí 5 - 40 V. K tomu účelu slouží napájecí konektor J1. Za tímto zdrojem následuje ochrana proti přepětí, přepólování, pojistka F1 proti zkratu a cívka L1 slouží pro zabránění vyzařování do napájecího zdroje.

Za tímto ochranným blokem jsou dva spínané napěťové regulátory. Výhodou spínaných regulátorů oproti klasickým stabilizátorům je menší velikost a nižší tepelné ztráty, tím pádem i menší nároky na chlazení. Prvním napěťovým měničem U1A má na výstupu napájecí napětí 5V potřebné především pro napájení bloku rozhraní CAN a převodníku 5V/3,3V. Za ním následuje regulátor U2B, který vytváří napájecí napětí 3,3V potřebné pro zbytek desky. Externí kondenzátory k těmto obvodům zobrazené na obrázku 5.2 slouží pro stabilizaci a vyhlazení napájecích napětí, cívky pak pro zabránění vyzařování. Na obrázku 5.9 je vidět indikace napájení pomocí diody DS4.

5.1.2 Procesor



Obrázek 5.3: Zapojení procesoru

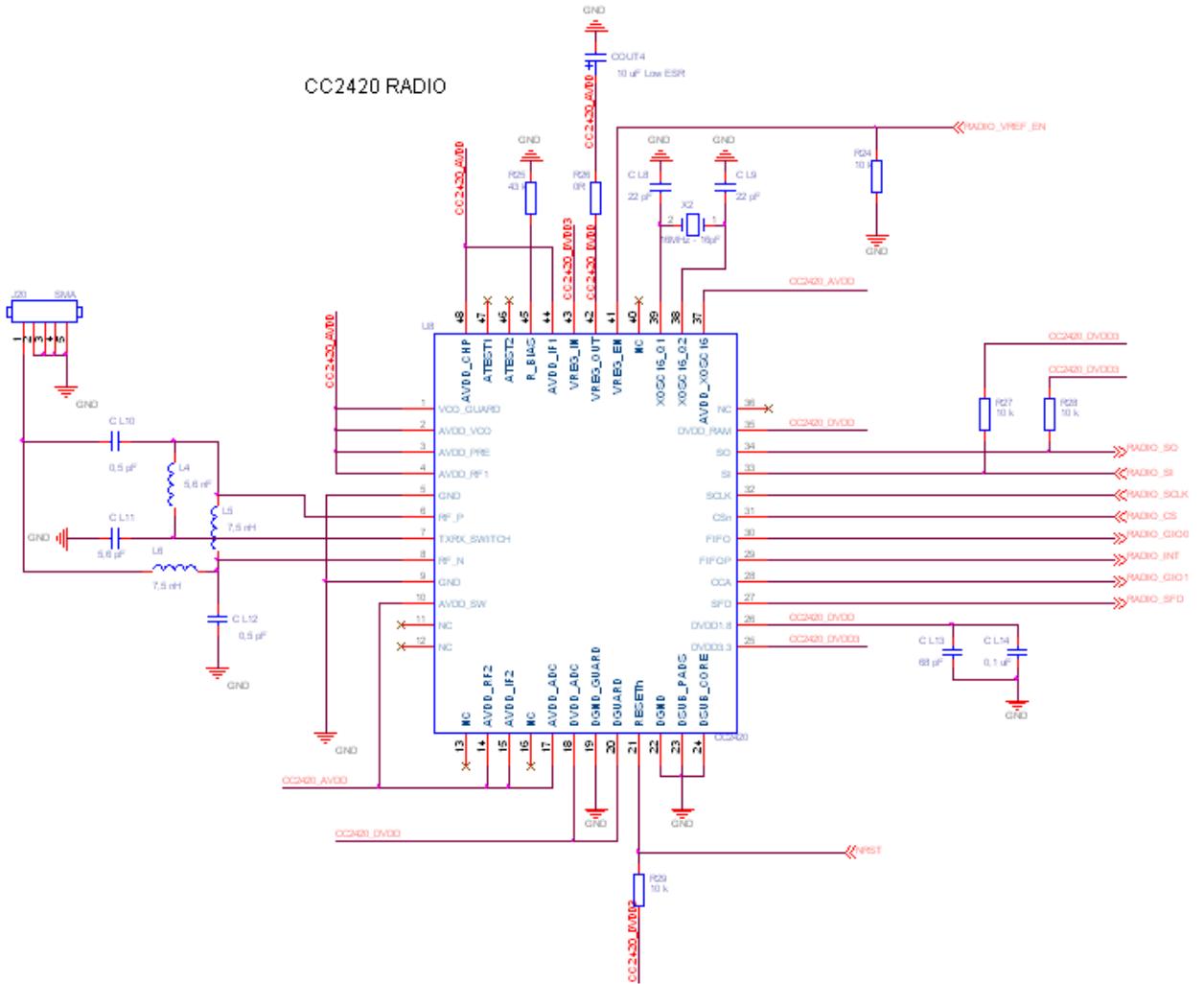
Řízení celého modulu má na starosti procesor AT91SAM7XC. Má 100 pinů a je v provedení pouzdra LQFP. Jeho výhodou je velká paměť flash pro aplikaci a tudíž nepotřebuje žádné externí paměti. Podrobnosti o tomto procesoru lze dohledat v (Atmel, 2006). Procesor má několik typů napájecích pinů. VDDIO, VDDFLASH, VDDIN, VDDOUT, VD-DCORE, VDDPLL. První tři z nich jsou připojeny k napětí 3,3V a slouží pro napájení vstupní výstupní logiky procesoru, napájení paměti flash a jako vstup pro napěťový měnič z 3,3V na 1,8V. Toto napětí je výstupem pinu VDDOUT a napájí piny VDD-CORE pro vnitřní logiku procesoru a VDDPLL pro napájení hlavního oscilátoru a programovatelného oscilátoru. Piny VDDIO a VDDFLASH jsou ve schématu značeny jako neviditelné.

K procesoru je připojen krystal X1 a spolu s kondenzátory CL1 a CL2 tvorí hlavní oscilační obvod. Další hodinový signál generuje vnitřní RC oscilátor a programovatelný oscilátor připojený na pin PLLRC. Pin NRST je vstupně/výstupní pin a slouží pro resetování procesoru, k tomuto účelu slouží tlačítko SW2 na obrázku 5.10. Zároveň lze tento pin nastavit na nulovou úroveň a tím resetovat ostatní moduly připojené k tomuto pinu. Pin ERASE po propojení propojky J8 vymaze vnitřní paměť flash, pin TST je použit pro vnitřní účely. Procesor má dále dva vstupně/výstupní porty PA a PB, které jsou

nastavitelné na požadovanou funkci. Každý z nich má 31 pinů a převážná část portu PA je navíc sdílená s piny pro programování flash paměti. Port B má poslední 4 piny sdíleny s analogovými vstupy AD převodníku. Některé tyto vstupy/výstupy jsem obsadil pro řízení připojených rozhraní a nejsou vyvedeny na univerzální konektory CON1, CON2, CON3 (znázorněny v blokovém schématu). Jedná se o ovládání rádiového modulu (9 pinů), LED diody (3 piny), uživatelské tlačítko (1 pin) a aktivace převodníku 5V/3,3V pomocí ENABLE_VC. Ten při nastavení na nulovou úroveň a propojení propojky J2 na obrázku 5.11 oddělí své vstupy a výstupy.

Dále jsou zde vyvedeny piny na konektor JTAG, pro aktivaci těchto pinů je třeba propojit propojku J15. Komunikaci přes rozhraní USB umožňuje dvojice DP a DM připojených k USB ovládacímu obvodu na obrázku 5.7.

5.1.3 Radiový modul



Obrázek 5.4: Připojení modulu ZigBee

Na obrázku 5.4 je znázorněn rádiový modul CC2420 od firmy Chipcon. Základní zapojení je uvedeno v datasheetu výrobce (Chipcon, 2006). Celý obvod je rozdělen na dvě části. Na analogovou, která má na starosti rádiové vysílání a na digitální, která komunikuje s řídícím procesorem. Obě tyto části jsou propojeny na pinu VREG_OUT přes 0-Ohmový rezistor R26. Modul má opět dvě napájecí části a to 3,3V, ze kterých je pak napětí regulováno přes vnitřní měnič na 1,8V.

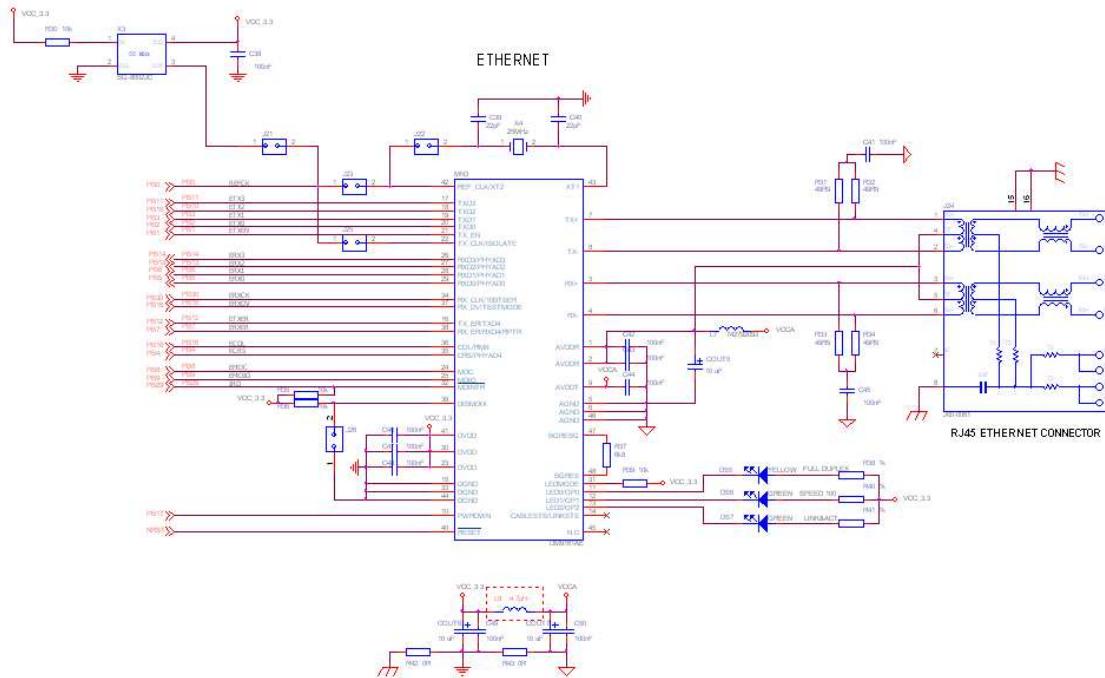
Připojení anténní části je převzato z datasheetu a liší se podle typu použité antény. Pro komunikaci s procesorem slouží rozhraní SPI a 6 digitálních vstupů výstupů. Jedním

z nich je resetovací signál připojený na pin NRST mikroprocesoru. Druhým důležitým pinem je RADIO_VREF_EN. Ten při nastavení na aktivní úroveň vypne napájení celého modulu. Ostatní digitální vstupy/výstupy slouží pro přenos a indikaci dat. CC2420 má svůj vlastní oscilační obvod zapojený dle datasheetu výrobce. Podrobné infomrace viz (Chipcon, 2006).

5.1.4 Ethernet

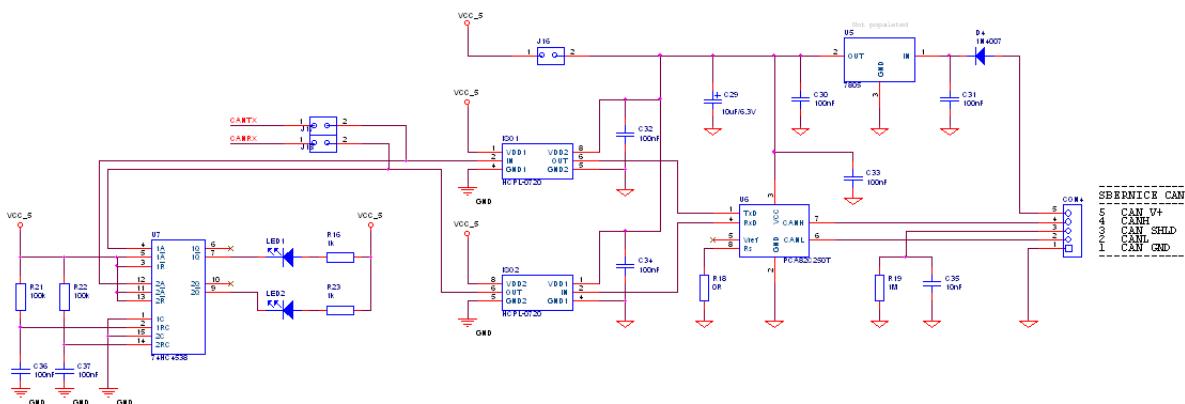
Pro připojení k Ethernetu má procesor implementovanou podporu EMAC (Ethernet Medium Access Control). To znamená, že je schopen komunikovat po sběrnici Ethernetu, je k tomu ale zapotřebí implementovat fyzickou vrstvu. To obstarává obvod DM9161AE, jehož zapojení lze vidět na obrázku B.3. Je spojen s mikroprocesorem přes 20 vstupů/výstupů, které musí být softwarově nastaveny pro komunikaci přes Ethernet. Je zde také přítomen pin NRST připojený na RESET vstup tohoto obvodu. Důležitým signálem je PB17, který umožňuje vypnout napájení pro tento modul. Dále jsou k tomuto obvodu připojeny dva oscilátory X3, X4 s různým hodinovým signálem. Přes propojky J21, J22, J23 a J25 jde libovolně nakonfigurovat hodinový signál na všechny hodinové vstupy. Obvod má přímo vyvedeny piny pro připojení LED diod indikujících stav na sběrnici.

Obvod je opět rozdělen na analogovou a digitální část. Tyto části jsou pak propojeny přes cívku L8. Základní zapojení tohoto obvodu lze najít v datasheetu (Davicom, 2005).



Obrázek 5.5: Připojení ethernetového rozhraní

5.1.5 CAN



Obrázek 5.6: Připojení sběrnice CAN

Procesor má v sobě integrovanou podporu pro komunikaci CAN. Pro připojení na reálnou sběrnici je však potřeba sběrnicový budič. Jeho schéma je zobrazeno na obrázku 5.6.

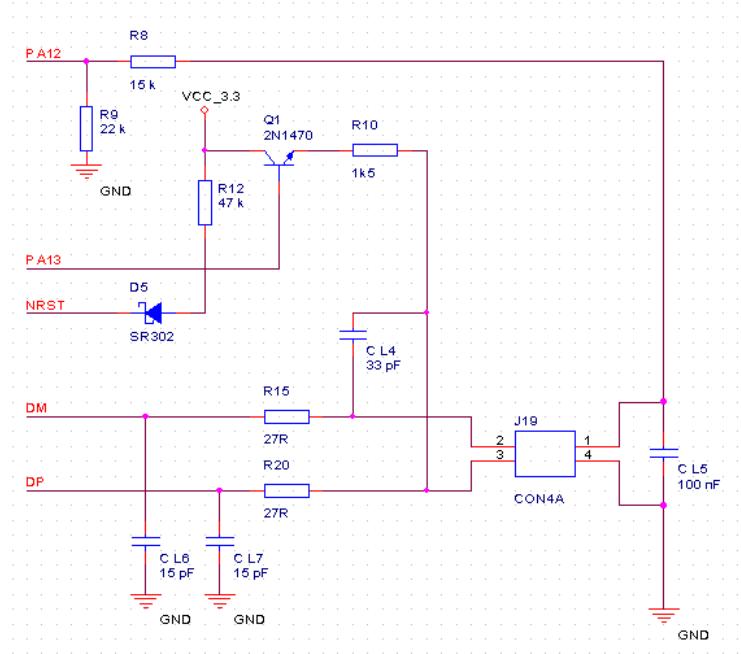
Základem je obvod s označením U6, který převádí signál z mikroprocesoru na signál na sběrnici CAN. Pro jeho napájení je potřeba napětí 5V. To lze dodat dvěma způsoby a to buď po propojení propojky J16 z hlavního napájecího zdroje desky a nebo po rozpojení propojky a připájení stabilizátoru U5 můžeme tuto část napájet přímo ze sběrnice CAN, kde je na konektoru vyvedeno napětí 12V. Pro galvanické oddělení této části slouží členy ISO1 a ISO2, které chrání obvod mikroprocesoru.

Dále pro indikaci dění na sběrnici CAN je zde použit obvod U7, což je monostabilní klopný obvod. Na jeho výstupech jsou připojeny LED diody, které zaznamenávají příjem a vysílaní dat. LED diody jsou propojeny z napájecího napětí 5V přes rezistory R16 a R23 k negovaným výstupům obvodu U7, které je po příchodu pulsu propojí na zem.

Protože CANTX a CANRX musí být nastaveny z universálních vstupů výstupů, jsou zde použity propojky J17 a J18 pro připojení/odpojení části CAN, aby mohly být piny mikroprocesoru využity pro jiný účel.

5.1.6 USB rozhraní

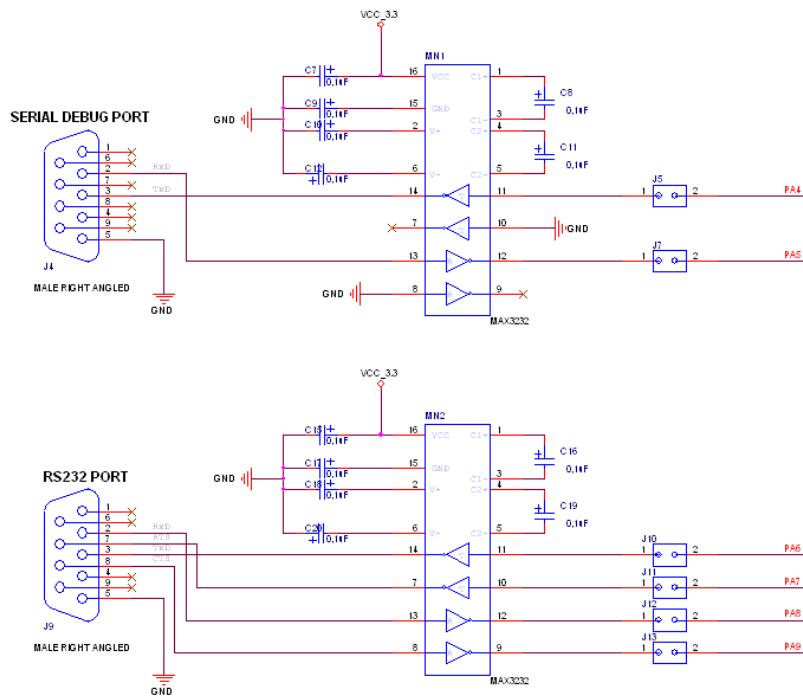
Testovací deska může být připojena k zařízení typu USB master pro seriovou komunikaci. Procesor má v sobě integrovaný transceiver a pro komunikaci jsou vyvedeny dva samostatné piny DDP a DDM. Na obrázku 5.7 je zapojení základního obvodu s vyvedeným konektorem. Pro detekci připojení slouží pin PA12 jako digitální vstup. Pin PA13 je digitální výstup a umožňuje připojit pull-up rezistor na DP. Podle napětí na tomto pinu pozná zařízení USB master, že je připojeno zařízení USB.



Obrázek 5.7: Připojení USB rozhraní

5.1.7 Seriová rozhraní k PC

Pro připojení k PC jsou na desce přítomny dva 9-ti pinové konektory typu DSUB, které mají přizpůsobeny napěťové úrovně. To zajišťují dva převodníky MAX3232, které jsou napájeny ze zdroje 3,3V. Pro komunikaci slouží vstupně výstupní porty mikroprocesoru nastavené na požadovanou funkci (seriová linka, debugging). K odpojení nebo připojení pinů slouží propojky J5, J7 a J10 až J13.



Obrázek 5.8: Zapojení seriových rozhraní

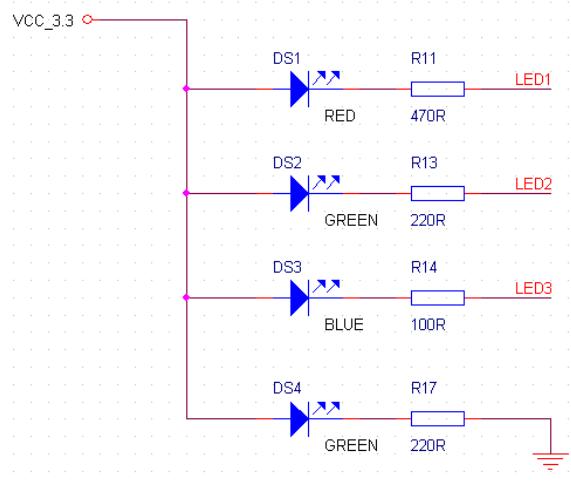
5.1.8 Rozšiřující konektory, uživatelské vstupy/výstupy

Aby programátor nebyl omezen pouze na komunikační rozhraní implementovaná na desce, jsou všechny vstupní i výstupní piny mikroprocesoru vyvedeny na 3 rozšiřující konektory CON1, CON2 a CON3. Tyto konektory mají definovaný počet vývodů a je zde přítomno 14 řídících signálů pro ovládání rozšiřujícího modulu GSM/GPRS z diplomové práce (Černý, 2005). Tyto řídící signály mají 5-ti voltovou logiku, proto je potřeba napěťového přizpůsobení.

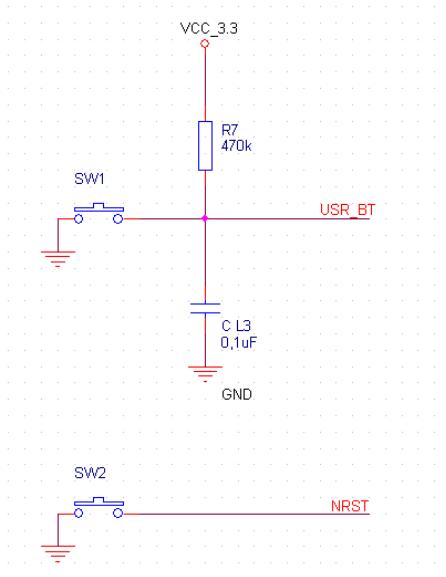
O to se stará obvod 74ALVC164245DL zobrazený na obrázku ??fig:prevodnik). Je rozdělen na stranu A a B, kde každá strana představuje je napájena různým napětím a (3,3V a 5V), kde toto napětí určuje napěťové úrovně signálů. Každá strana má dva porty, které mohou být nakonfigurovány pomocí ovládacích vstupů 1OE, 2OE, 1DIR, 2DIR na vstupní nebo výstupní. V našem případě jde port 1 ze strany B do A a port 2 naopak. Pokud nastavíme signál ENABLE_VC na nulovou úroveň budou strany galvanicky odděleny.

Na obrázcích 5.9 a 5.10 jsou vidět diody, které lze libovolně použít při programování

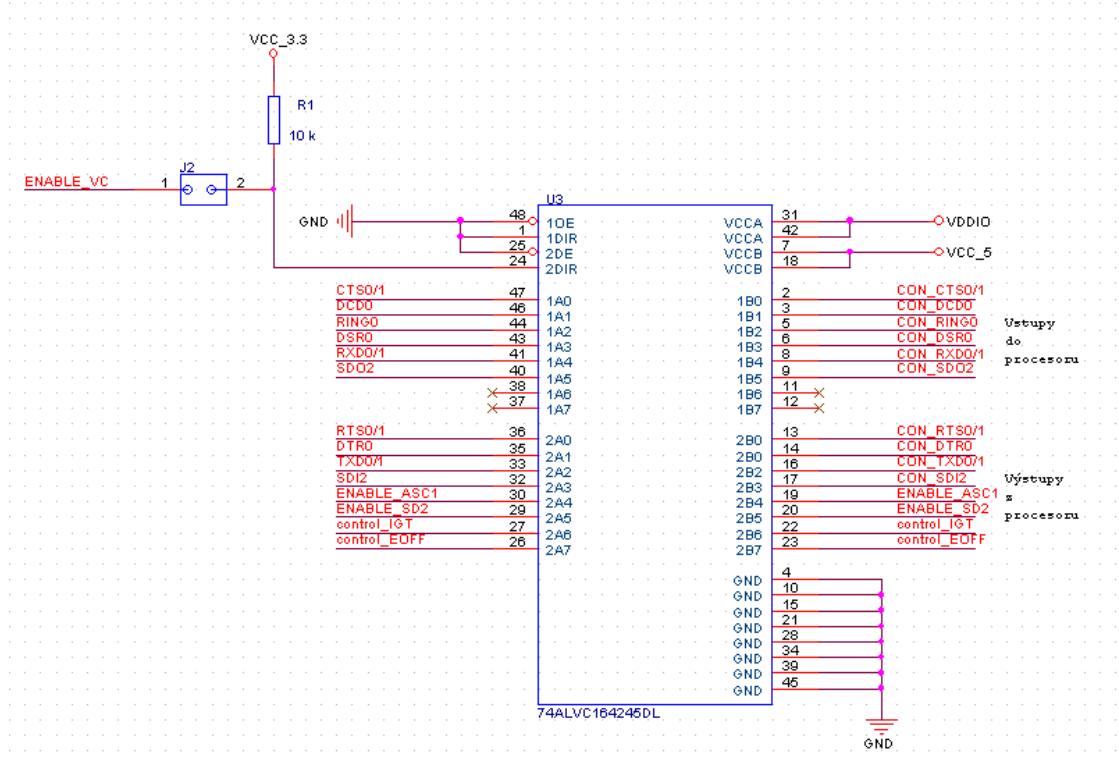
(kromě diody DS4, která indikuje vstupní napájení), jedno uživatelské tlačítko a jedno tlačítko pro reset mikroprocesoru.



Obrázek 5.9: Indikační LED diody



Obrázek 5.10: Tlačítka



Obrázek 5.11: Zapojení převodníku 5V/3,3V

Kapitola 6

Závěr

Tato práce si dala za cíl udělat rešerši na poli aktivní bezpečnosti v automobilovém průmyslu a na základě průzkumu navrhnut vlastní systém, který bude chránit řidiče varovným signálem ještě před nehodou.

Navrhli jsme komunikační protokol pro bezdrátový přenos, který se opírá o statistická měření v praxi a provedli jsme vlastní simulace na PC. Protokol byl navržen pomocí stavových automatů ve vývojovém prostředí *UPPALL*. V tomto prostředí jsme také ověřili správnost návrhu modelu pomocí verifikačních formulí.

Dále jsme provedli průzkum bezdrátových technologií a hardwarových prostředků a vybrali jsme komunikační a řídící modul Tmote Sky. Na základě tohoto výběru jsme se seznámili s operačním systémem TinyOS pro bezdrátové senzorové platformy a programovacím jazykem nesC pro tento operační systém.

Souběžně se studiem TinyOS jsme vypracovali krátký dokument, který navrhoval možnosti určování směru a případně polohy vozidel. Na základě tohoto článku jsme se rozhodli pro použití systému GPS na určení polohy a pro použití akcelerometru na detekci nebezpečného chování vozidla (prudké brzdění, smyk) a navrhli jsme rozšiřující hardwarovou desku pro Tmote Sky. Tu jsme následně vyrobili a spolu s moduly otestovali.

Pro tento hardware jsme za pomoci TinyOS navrhli aplikaci, která má za úkol přenášet bezdrátově data o poloze a zrychlení z jednoho přípravku do druhého a následně zobrazovat přijatá data v PC.

Nakonec jsme se rozhodli, že celý systém rozšíříme o komunikaci přes sběrnici CAN v automobilu a to z toho důvodu, aby byla umožněna lepší spolupráce s ostatními podpůrnými prvky aktivní bezpečnosti ve vozidle. K tomuto účelu jsme využili předchozí diplomové práce (Musil, 2003) a (Černý, 2005) a navrhli hardwarové schéma testovací

řídící jednotky s použitím modernějšího mikroprocesoru. Na desku jsme také implementovali modul ZigBee pro bezdrátovou komunikaci. Zároveň byla zachována hardwarová kompatibilita pro rozšiřující modul staré řídící jednotky.

Celkově lze říci, že jsme byli při návrhu úspěšní. Navržený protokol nevykazoval žádné chyby, ale z výpočetních důvodů byl ověřen pouze pro tři komunikační jednotky. Ze simulací v článku (Yang X., n.d.) však nepředpokládáme, že by se systém choval ve skutečnosti jinak. Navržené hardwarové prostředky představují dostatečnou základnu pro další testování. Programování testovací aplikace bylo náročnější především díky nedostačující dokumentaci pro TinyOS a jazyk nesC. Nakonec se však povedlo přenést bezdrátově data o poloze i zrychlení. Bohužel aplikace v PC nezobrazuje data o zrychlení v grafické podobě a bylo by potřeba se tomuto více věnovat.

S přibývajícími znalostmi o problematice se částečně měnily požadavky na navrhovaný systém a definovaly se nové cíle. Tato práce připravila hardware pro automobilový varovný systém a na aplikaci ověřila ve skutečnosti měření polohy a zrychlení. Navržený hardware je cenově mnohem dostupnější než související světové projekty, což bylo jedním z požadavků, a mohl by v budoucnu doplňovat připravované standardizované systémy.

Literatura

- Analog (2002), *ADXL210E - Dual Axis Accelerometer with Duty Cycle*, Analog Devices.
Rev. 0.
- Atmel (2006), *AT91SAM7XC Microcontroller datasheet*, Atmel.
- Behrmann, G. (2004), *A tutorial on Uppaal*.
- Brown, A. C. (n.d.), *Vehicle to Vehicle Communication Outage and its Impact on Convoy Driving*.
- Chipcon (2006), *CC2420 radio module datasheet*, Chipcon.
- Daid, C. M. (n.d.), *GPS Overview*.
<http://www.palowireless.com/gps/tutorial.asp>.
- Davicom (2005), *DM9161A 10/100 Mbps Fast Ethernet datasheet*, Davicom Semiconductor, Inc.
- Černý, M. (2005), *Komunikace CAN v automobilu*, ČVUT FEL. Diplomová práce.
- EU (2007), PReVENT, European Union Project.
<http://www.prevent-ip.org/>.
- Fal (2006), *JP13 Family Datasheet*, Falcom. rev. 1.04.
- Levis, P. (2006), *TinyOS programming*. Revision 1.2.
- Mote (2006), *Tmote Sky datasheet*, Moteiv Corporation.
- Musil, M. (2003), *Operační systém reálného času pro automobily*, ČVUT FEL. Diplomová práce.
- Schulze, M. (n.d.), *PReVENT: A European program to improve active safety*. Article.

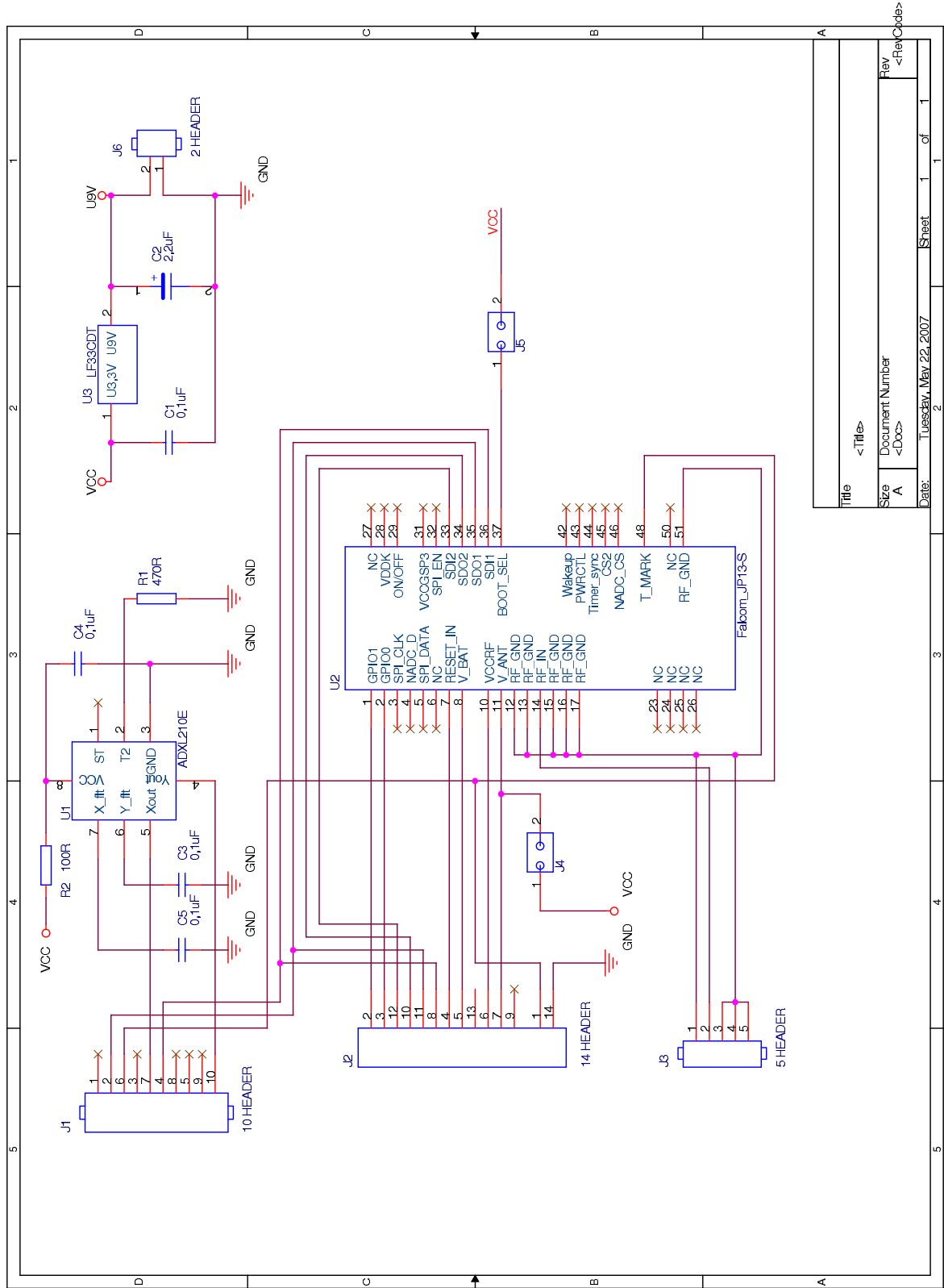
- SiRF (2005a), *NMEA Reference Manual*, SiRF Technology, Inc.
- SiRF (2005b), *SiRF Binary Protocol Reference Manual*, SiRF Technology, Inc.
- TEP files* (2007), in ‘Online TinyOS Documentation’, TinyOS Developement Groups.
<http://.tinyos.net>.
- TI (2006), *MSP430x16x Mixed Signal Microcontroller datasheet*, Texas Instruments.
- Vojáček, A. (2006), *ZigBee - novinka na poli bezdrátové komunikace*, Internet source. HW server.
- Wikipedia (2007), *Global Positioning System*.
http://cs.wikipedia.org/wiki/Global_Positioning_System.
- Yang X., J. L. (n.d.), *A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning*.
http://research.microsoft.com/~zhao/pubs/yang_x_v2v.pdf.

Příloha A

Elektronické schéma GPS/ACC modulu

PŘÍLOHA A. ELEKTRONICKÉ SCHÉMA GPS/ACC MODULU

II

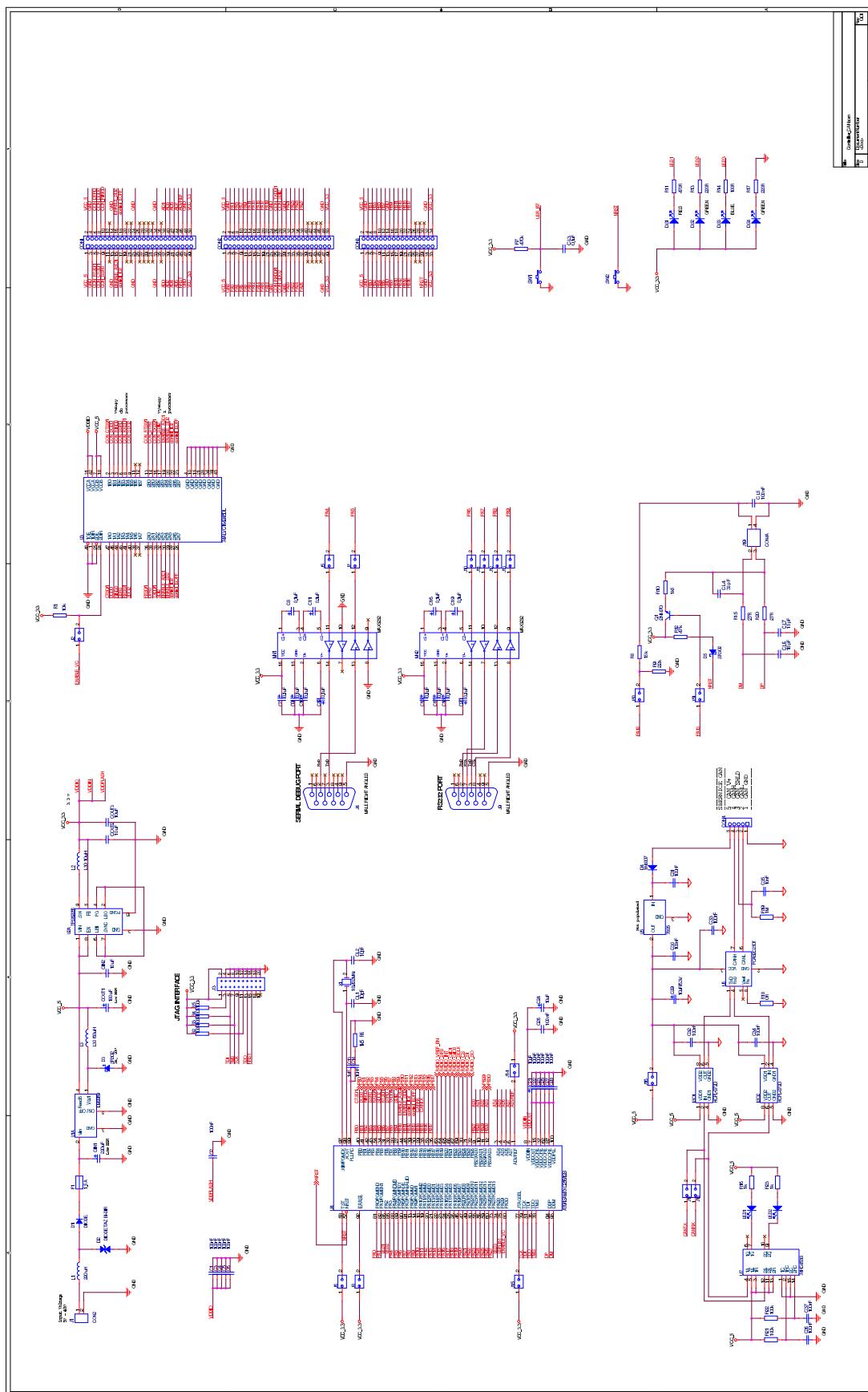


Obrázek A.1: Schéma navrhnutého modulu GPS/ACC

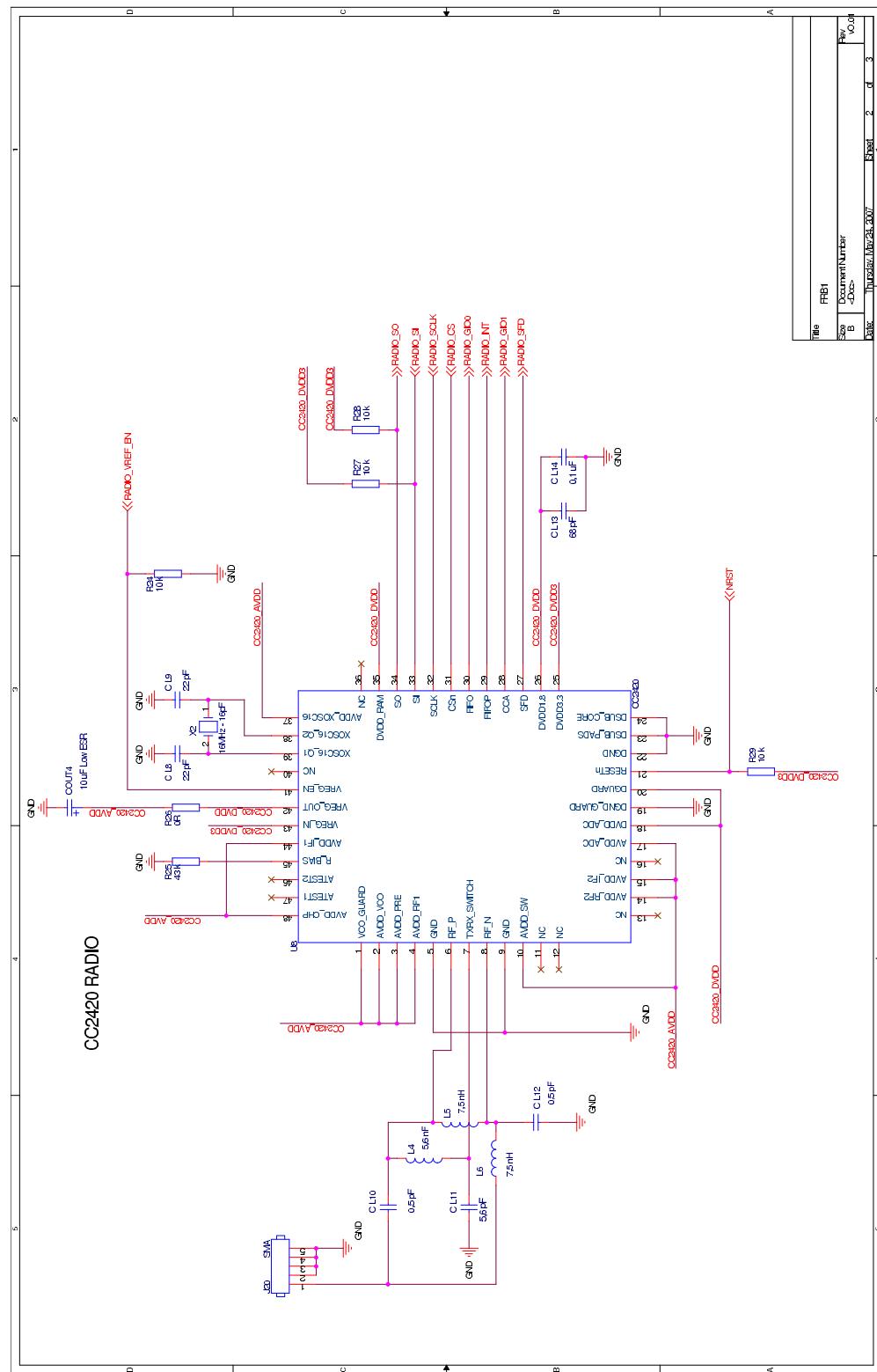
Příloha B

Elektronické schema rozšiřující Řídící Jednotky

PŘÍLOHA B. ELEKTRONICKÉ SCHEMA ROZŠIŘUJÍCÍ ŘÍDÍCÍ JEDNOTKY IV

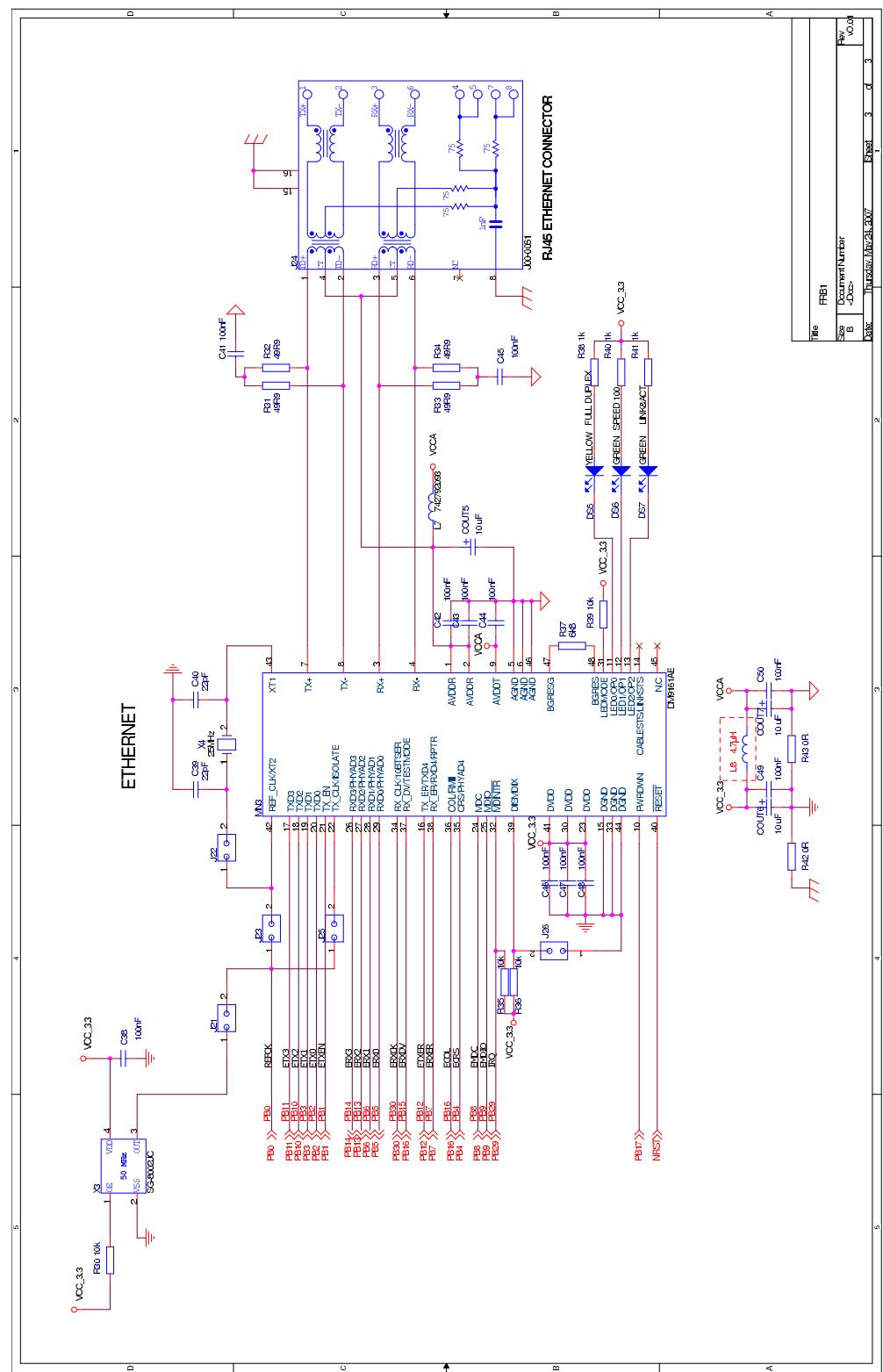


Obrázek B.1: Schéma mikroprocesorové části



Obrázek B.2: Schéma rádiového modulu

PŘÍLOHA B. ELEKTRONICKÉ SCHEMA ROZŠIŘUJÍCÍ ŘÍDÍCÍ JEDNOTKY VI

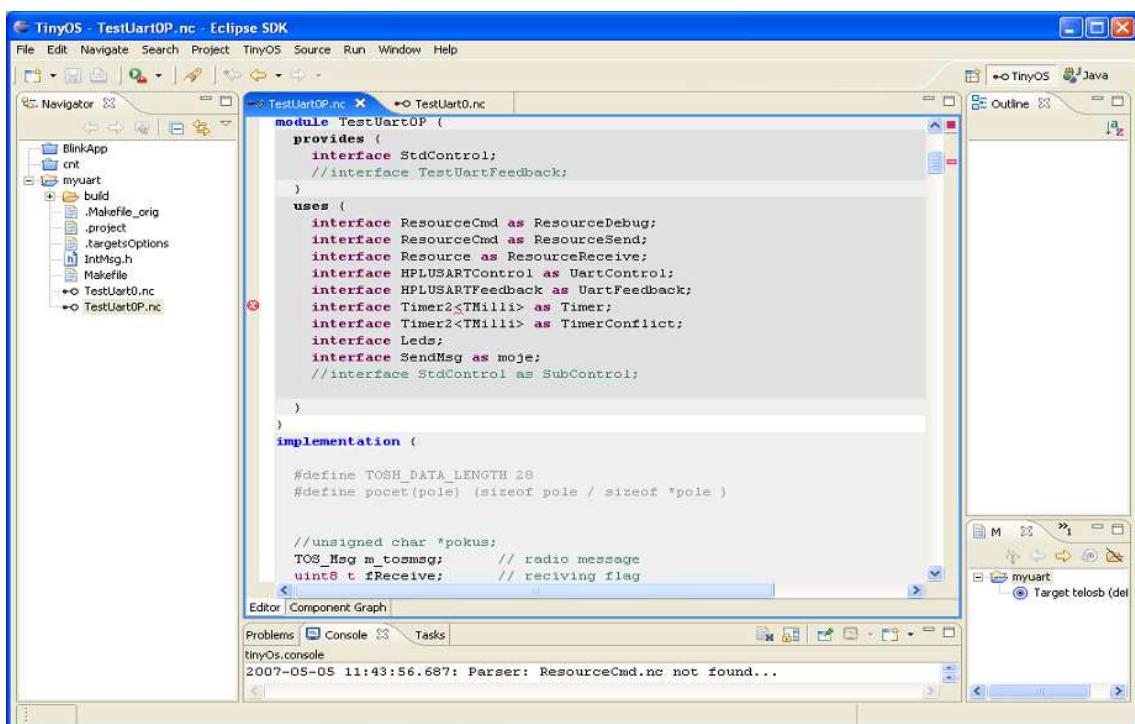


Obrázek B.3: Schéma ethernetového budiče

Příloha C

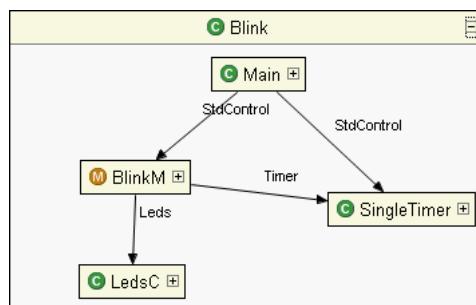
Praktické připomínky k prostředí Eclipse

V kapitole 4.1.3 jsem se zmínil o programovacím prostředí Eclipse. Tento nástroj původně vznikl pro usnadnění kódování jazyka Java. V dnešní době podporuje mnoho dalších programovacích jazyků, jedním z nich je pak jazyk nesC pro operační systém TinyOS. Na obrázku C.1 je zobrazeno vývojové prostředí aplikace Eclipse pro TinyOS programování.



Obrázek C.1: TinyOS plugin pro Eclipse

Výhodou tohoto prostředí je bezpochyby i zvýrazňování chybové syntaxe a znázornění struktury projektu. Tu lze zároveň zobrazit i v přehledných diagramech. Na obrázku C.2 je vidět graf jednoduché aplikace blikání LED diody. Modul *Main* přes rozhraní *StdControl* ovládá moduly *SingleTimer* a *BlinkM*, kde druhý zmíněný se váže na konfigurační soubor nižší vrstvy *LedsC*. Všechny komponenty lze v grafu i rozvinout a zobrazit tak jejich podrobnější vazby a vnitřní strukturu, která je mnohem obsáhlější.



Obrázek C.2: Graf aplikace v prostředí Eclipse

Na druhou stranu musím říci, že tato varianta doposud není až tak propracovaná, jak by se mohlo zdát, protože ve verzi TinyOS pluginu 0.5.9 je Eclipse nestabilní a dochází k občasnému „zaseknutí“ nebo přestanou fungovat některé funkce a je třeba ho restartovat. Zároveň je možno si při instalaci vybrat z několika možností, a to použít již připravenou instalaci Cygwinu a TinyOS, nebo spolu s instalací pluginu instalovat i verzi *TinyOS 1.1*, nebo obě varianty, kdy bude mít Eclipse k dispozici obě verze. Bohužel i při správném nastavení všech parametrů se mi nepodařilo spustit vytvořenou aplikaci v mé instalaci *TinyOS Boomerang 2.0.4* přímo z vývojového prostředí Eclipse. Je potřeba bud' nakopírovat aplikaci z pracovního adresáře Eclipse do struktury Cygwinu a přeložit v příkazové řádce nebo použít vestavěnou verzi v pluginu ve verzi *TinyOS 1.1*. Zde ale není podpora pro platformu Tmote. Je možno použít starší platformu telosb, což je její předchůdce, ale některé aplikace nebudou přeloženy, protože zde došlo k některým hardwarovým změnám. V budoucí verzi pluginu je plánováno rozšíření i pro podporu hardwarových modulů Tmote a také podpora pro *TinyOS 2.0*.

Příloha D

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy.

- **Protokol/Uppaal/**: Návrh komunikačního protokolu, data pro UPPAAL, nástroj UPPAAL
- **HW/GPS_ACC/**: Návrh plošného spoje GPS/ACC desky, data ve formáto OrCAD
- **HW/CAN CU/**: Návrh plošného spoje Řídící Jednotky, data ve formáto OrCAD
- **SW/GPIO/**: Aplikace pro čtení dat z akcelerometru, zdrojový kód pro TinyOS
- **SW/GPS/**: Aplikace pro čtení GPS dat, zdrojový kód pro TinyOS
- **SW/SocketClient/**: Aplikace pro zobrazení přijímaných dat na PC, zdrojový kód pro MSV2005
- **Dokumentace/**: Diplomová práce ve formátu pdf.
- **Dokumentace/Datasheet/**: Datasheetové listy k obvodům v návrhu ve formátu pdf.