České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky

Bezdrátová síť pro monitorování pacientů trpících Parkinsonovou chorobou II.

PRAHA 2009 Erik Markovič

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

Abstrakt

Tato diplomová práce se zabývá bezdrátovou senzorovou sítí pro monitorování stavů pacientů s Parkinsonovou chorobou.

Bezdrátová senzorová síť je navržená v topologii hvězda s komunikačním protokolem Master – Slave. Síť se skládá z pěti zařízení platformy TelosB. Jedno zařízení plní úkol jednotky master a slouží ke sběru dat z jednotek slave a k řízení komunikace s jednotkami slave. Další čtyři zařízení slouží k odběru dat z přídavného modulu s akcelometrem, v komunikačním protokolu jsou řazeny mezi jednotky slave.

Po prostudování prostředí TinyOS-2.x, zařízení platformy TelosB a přídavných zařízení s akcelometry byly vytvořeny ovladače pro přídavné moduly. Poté byl navržen komunikační protokol a následně jeho implementace s vizualizací.

Pro práci se zařízením byl zvolen linux Xubuntu 2.0 s TinyOS. Tento operační systém podporuje prostředí TinyOS, ve kterém se programuje pomocí programovacího jazyka NesC. K vytvoření aplikace pro vizualizaci a ukládání dat byl použit programovací jazyk Java.

Abstract

This diploma thesis deals with wireless sensory network used for monitoring the condition of Parkinson disease patients.

Wireless sensory network is designed as a star topology using Master-Slave communication protocol. The network consists of five TelosB platform devices. One of the devices acts as the master and collects data from the slave devices (units). The remaining four slave devices collect data from the auxiliary accelerometer module.

After a study of the TinyOS-2.x environment, TelosB platform and auxiliary accelerometer modules, the module's drivers were created. In addition, a communication protocol was designed along with its implementation and visualization.

The Linux branch Xubuntu 2.0 with TinyOS was chosen as the operating system (OS). This OS supports TinyOS environment that uses NesC as the programming language. The visualization and PC-based data collecting was developed in Java programming language.

Obsah

1. Úvod

V současné době jsou při pozorování pacientů s Parkinsonovou nemocí používána zařízení, která jsou tvořena komplikovanou soustavou kabelových spojení. Nevýhodou tohoto řešení jsou rozměry a omezení mobility pacienta, kdy pacient musí během měření setrvat na jednom místě. Tyto omezení pak vyžadují zkrácení doby sběru dat z pozorování pacienta.

V diplomové práci se zabývám bezdrátovou senzorovou sítí, kterou by následovně bylo možné použít pro pozorování pacientů s Parkinsonovou nemocí. Na rozdíl od stávajícího systému, který stojí na kabelové síti, neklade bezdrátová síť omezení na mobilitu a snižuje zátěž pacienta během pozorování. Jako další výhody bezdrátového řešení lze uvést lepší přesnost měření, nižší energetickou náročnost a malé rozměry. Malé rozměry a bezdrátové spojení nám dávají možnost pozorovat pacienta v delším časovém horizontu, aniž bychom pacienta příliš omezovali během sběru dat. Použitím bezdrátové technologie je eliminována kabeláž, která mohla být zdrojem zkreslení během měření. Nevýhodou tohoto řešení je nutnost zajistit energii pro bezdrátové senzory, řešit úsporné režimy, kontrolu stavu baterií apod.

Při návrhu bezdrátové sítě jsem vycházel z předchozí práce "Bezdrátová síť pro monitorování pacientů trpících Parkinsonovou chorobou" [1], která pracovala nad starším operačním systémem TinyOS 1.x a používala komunikační protokol Open-ZB Stach. Ve své diplomové práci jsem přešel na vyšší verzi operačního systému TinyOS 2.x. Původní komunikační protokol jsem nahradil vlastním, ve kterém jsem řešil problematiku snížení množství přenesených dat a tím zkrátil dobu potřebnou k jejich přenesení. Hardwarová platforma pro bezdrátový přenos dat vychází z původního systému Tmote Sky od společnosti MoteIV s chipem ZigBee firmy Chipcon. Tato platforma podporuje operační systém TinyOS.. Tento systém byl rozšířen o moduly osazené akcelometry (MMA7260Q a LIS3LV02DQ).

Pro rozšiřující moduly s akcelometry jsem vytvořil ovladače pro operační systém TinyOS 2.x, které zajišťují přenos dat z akcelometrů do modulu Tmote Sky. V dalším kroku jsem navrhl a implementoval komunikační protokol Master - Slave, který přenáší získané data z akcelometru pomocí bezdrátové sítě do hlavního modulu Tmote Sky. Z hlavního modulu jsou pomocí rozhraní USB přenesena data do PC, kde jsou následně zpracovaná pomoci aplikace napsané v jazyce Java. Tato aplikace tvoří uživatelské rozhraní pro konfiguraci zařízení, vizualizaci dat a ukládaní naměřených dat do souboru.

V průběhu realizace jsem se zaměřil na modul s akcelometrem MMA7260Q, jelikož s akcelometrem LIS3LV02DQ by se muselo v rámci komunikačního protokolu řešit přepínání sběrnice, které by omezovalo využití bezdrátové komunikace.

V práci se snažím splnit požadavky na rychlý přenos dat, komprimaci dat, optimalizaci komunikačního protokolu s co nejmenším dopadem na vzorkovací frekvenci akcelometru. Z hlediska spotřeby jsem se zaměřil na možnost úsporného režimu, případné uspání senzoru.

1.1. Parkinsonová nemoc

Tuto nemoc poprvé v roce 1817 popsal londýnský lékař James Parkinson. Původ nemoci není znám. Dochází k předčasné poruše funkce a poškození struktur v hloubi mozkových polokoulí, která se nazývají basální ganglia. Jde o seskupení nervových buněk do skupin, které jsou mezi sebou vzájemně propojeny [2].

V lidském mozku jsou struktury, které ovlivňují pohybové aktivity a právě Parkinsonova choroba je nemoc, která postihuje mozkové systémy ovlivňující automatické pohyby, svalové napětí a koordinaci pohybů. Příznaky bývají zpočátku vcelku nenápadné. Nemocný postřehne zvýšenou únavnost, pocity zatuhnutí, křeče, pocity tíže končetin, někdy dokonce bolesti kloubů a zad (nepřímo zapříčiněné tuhostí), celkové zpomalení a poruchy chůze. V jiných případech může na nemoc upozornit téměř výhradně klidový, rytmický, pomalý třes na jedné z horních končetin připomínající počítání peněz nebo hnětení kuličky. Jak nemoc pokračuje, setkáváte se s jejími dalšími hlavními příznaky. Pocity tuhosti, jednostranným, později oboustranným klidovým třesem, pomalostí a chudostí pohybů. Nápadný je zejména maskovitý obličej s nedostatečnou mimikou, charakteristická šouravá chůze v předklonu s rukama ohnutými v loktech, postupně se přidávají poruchy rovnováhy, časté jsou pády, zácpa, zvýšený mazotok a poruchy močení. Vývoj nemoci trvá měsíce až roky. Neléčené onemocnění obvykle končí po několika letech celkovou nehybností. Naštěstí v dnešní době již existují prostředky, jak můžete tuto nemoc výrazně zpomalit a stabilizovat natolik, aby byl postižený člověk schopen žít plnohodnotný život a vykonávat své zaměstnání [2].

Vlastní podstata nemoci spočívá v nedostatku dopaminu. To je přenašeč signálů mezi nervovými buňkami. Jeho nedostatek se projeví právě v poruše mozkových okruhů, které regulují mimovolní svalové napětí, provádění pohybů a jejich automatičnost. Navíc má i vliv na psychické a další funkce, např. zažívání. Tento nedostatek se lečí pomocí léku, který se volí pomocí určitého klíče, jakými jsou např. intenzita příznaků, věk pacienta či jeho přídatné nemoci [2]. A zde se dostáváme k nezbytnosti monitorování pacienta, pro jeho léčbu.

1.2. Monitorování příznaků pacientů

Od 90. let se používají malé elektronické zařízení pro monitorování lidské aktivity. Tyto zařízení nazýváme ActiGraphs a skládají se z inerčních senzorů (akcelometrů nebo gyroskopu), filtrů, paměti, rozhraní a číslicových obvodů.

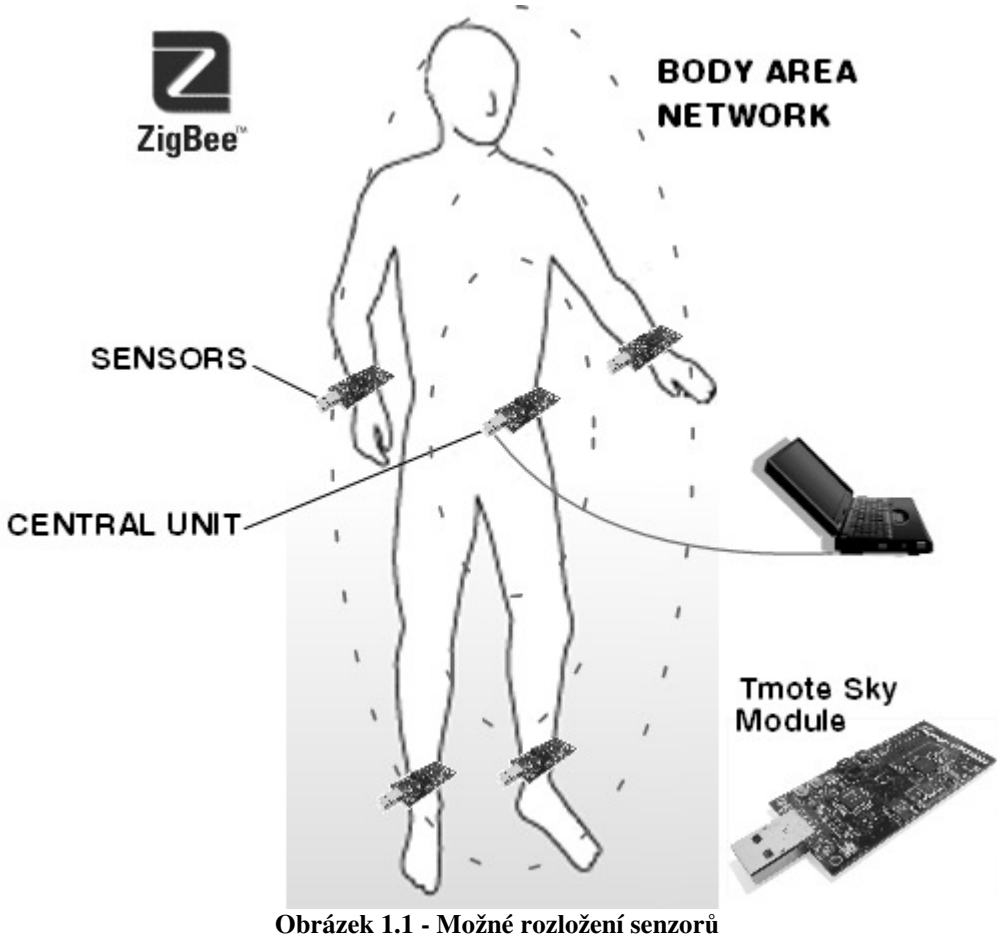
K typickým projevům Parkinsonové choroby patří třes a nepřirozený pohyb. Tyto projevy se objevují periodicky a je nutné použít několik snímačů umístěných na těle, aby byly zjištěny všechny příznaky. Proto se senzory připevňují obvykle k pacientovu zápěstí, kotníku, kolenu, zádům nebo hrudníku a zaznamenává se jeho aktivita. Velkou nevýhodou je omezující kabeláž, která vede ze senzorů umístěných na těle pacienta. Tato kabeláž svazuje pacienta na jednom místě, což pro něho není příjemné, proto je bezdrátová senzorová síť pro pacienty vhodnější [1].

1.3. Bezdrátová senzorová síť

Je to bezdrátová síť, která se skládá z prostorově distribuovaných autonomních zařízení, užívajících senzorů na sledování prostředí, jako například teplota, zvuk vibrace, tlak, pohyb nebo znečišťující látky na různých místech. Vývoj bezdrátových senzorových sítí byl původně motivován vojenskými aplikacemi (např. sledování bojiště). Nicméně se bezdrátové senzorové sítě začaly používat v mnoha průmyslových a civilních oblastech, včetně sledovaní průmyslových procesů, kontroly a stavů přístrojů, v monitorování životního prostředí a přírodních stanovišť, v zdravotních aplikacích, v domácích automatizacím atd. [3].

Bezdrátový snímač (node nebo motes) se obvykle skládá z čidla, bezdrátového komunikačního zařízení, malého mikrokontroleru a zdroje energie, kterým je obvykle baterie. Náklady na čidla se pohybují v rozmezí pár korun, až několika tisíc korun v závislosti na velikosti senzorové sítě a složitostí požadovaných jednotlivých senzorových uzlů. Velikost a omezení nákladu na senzor má odpovídající omezení na zdroje jako energie, paměť, výpočetní rychlost a šířka pásma [3].

Na obr. 1.1 můžete vidět rozložení senzorů v mé práci.



Obrázek 1.1 - Možné rozložení senzorů

2. Hardware

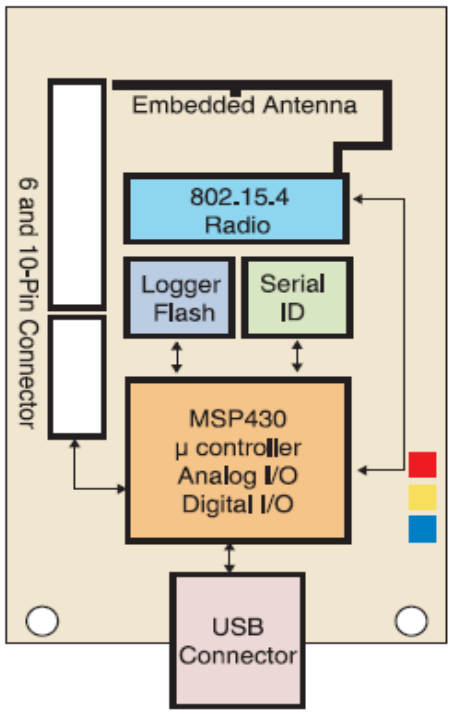
2.1. TelosB

Jako platformu pro bezdrátovou senzorovou síť používám Tmote Sky od firmy MoteIV nebo Crossbow Telos od firmy Crossbow. Jde o shodná zařízení lišící se pouze výrobcem. Společné označení této platformy je TelosB. Katedra řídící technicky disponuje oběma zařízeními, a proto bylo toto zařízení vybráno.

Platforma má open-source operační systém TinyOS. Má nízkoenergetické napájecí bezdrátový modul, navrženy tak, aby umožnila experimentování v bezdrátové komunikaci. TelosB disponuje vlastnostmi potřebnými pro laboratorní studium: USB programovací schopnosti, IEEE 802.15.4 kompatibilitu, vysokou rychlost přenosu dat, integrovanou anténu, nízké napájení MCU s rozšířenou pamětí. [4]

2.1.1. Technické parametry

- 250kbps 2.4GHz IEEE 802.15.4 CC2420 Chipcon Wireless Transceiver
- 8MHz Texas Instruments MSP430F1611 Chipcon Wireleess Transceiver
- Integrovaný ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrovaná anténa s 50m dosahem v uzavřených prostorech / 125m dosahem venku
- Integrované čidlo vlhkosti, teploty a senzor světla
- Nízká spotřeba proudu a rychlý přechod ze spánku
- Programovaní a datové spojení přes USB port
- Šestnácti pinový rozšiřující konektor a SMA anténní konektor



Obrázek 2.1 - Blokový diagram TelosB

2.1.2. Popis modulu TelosB

Zařízení napájí dvě baterie typu AA s provozním napětím 2,1 až 3,6 V nebo USB port s provozní napětím 3V. USB port lze použít k programování zařízení a komunikací s PC.

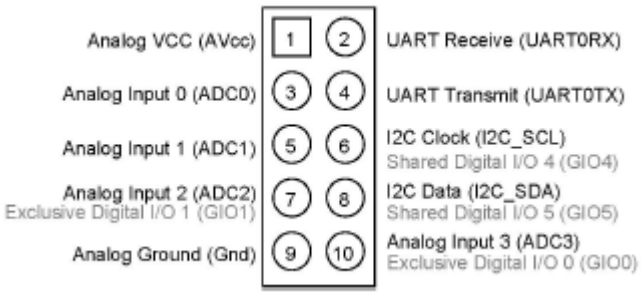
Pro bezdrátovou komunikaci TelosB využívá Chipcon CC2420 IEEE 802.15.4, který poskytuje PHY, některé MAC funkce a programovatelný výstupní výkon. Tranceiver standardně komunikuje ve volném pásmu 2.45 GHz, přesněji v 16 kanálech s přenosovou rychlostí 250 kbit/s v rozmezí frekvencí 2.4 GHz až 2.4835 GHz. Specifikaci najdete v datasheetu od Chipcon [5]. K řízení je použit 16bitovým RISC mikrokontrolér TI MSP430 s programovatelnou Flash pamětí 48KB a 10KB RAM, který využívá ke komunikaci SPI (Serial Port Interface) port a řady digitálních I/O kanálu a přerušení. Komunikace v rozšiřujícím konektoru je realizována pomocí I2C (Inter IC) a 12bitovým ADC převodníkem. [6]

Další součástí TelosB je Flash paměť STM25P80 o velikosti 1 MB, která může být snadno použita pro trvalé ukládání dat, kódu a dalších informací do modulu TelosB, dokud nedojde k jejímu přemazání. Paměť flash sdílí komunikační kanál SPI s vysilačem CC2420 a externím konektorem SPI. V případě současného používání je důležitá dobrá arbitrace na sdílené sběrnici. [1]

TelosB používá pro řízení USB řadič FTDI chip od Future Technology Devices International ltd. Aby bylo možné zařízení použít, je nutné nainstalovat ovladače, které jsou volně dostupné a k dispozici jsou na stránkách výrobce [7]. TelosB se chová ve Windows jako COM port a v Linuxu jako zařízení. K počítači lze najednou připojit více zařízení každému je přiděleno číslo COM portu ve Windows, případně v Linuxu dostane svoji adresu zařízení.

K dispozici jsou 3 LED diody, které jsou ovládaný uživatelem, pomoci jím vytvořeného programu. Dále nabízí slot pro připojení externí antény, která může zvýšit dosah zařízení z 50 metrů na 125 metrů [8].

Zařízení disponuje dvěma rozšiřujícími konektory, které lze využit k připojení dalších zařízení, která mohou řídit nebo rozšiřovat modul (senzory, digitální periferie). 10 pinový konektor je na obr. 2.2 a 6 pinový konektor na obr. 2.3.



Obrázek 2.2 – 10 pinový rozšiřující konektor

Obrázek 2.3 – 6 pinový rozšiřující konektor

2.2. Přídavný modul s akcelometrem

Z předchozí práce [1] jsem použil dva přídavné moduly s akcelometrem. Jeden z přídavných modulů má zabudovaný analogový akcelometr MMA7260Q od firmy Freescale Semiconductor. Druhý přídavný modul má zabudovaný digitální akcelometr LIS3LV02DQ od firmy ST Microelectronics. Přídavné moduly byly zvoleny s ohledem na požadavky měření (vzorkovací frekvence, spotřeba, citlivost, atd.).

2.2.1. Přídavný modul s digitálním akcelometrem LIS3LV02DQ

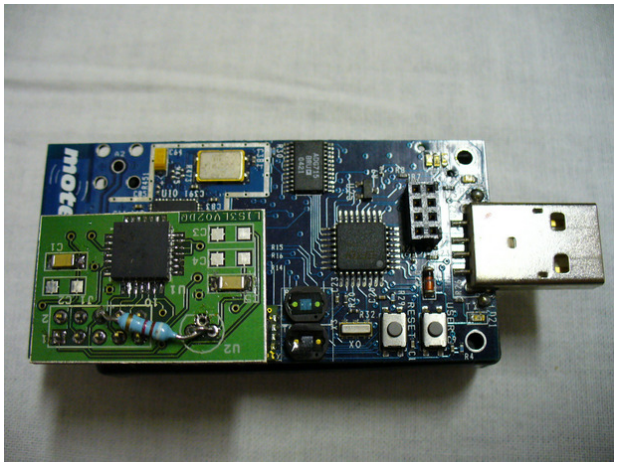
Akcelometr je nastavitelný na 2g nebo 6g, má tříosý digitální výstup lineárního zrychlení. Na IC rozhrání je modul schopen přijmout informace ze senzoru a poskytnout naměřené zrychlení dalšímu zařízení pomocí I2C nebo SPI rozhraní [8].

Jelikož TelosB nabízí v hlavním přídavném 10 pinovém konektoru jen dva I2C piny z I2C sběrnice, musím využít propojení pomoci I2C sériového rozhrání, protože SPI nelze použít.

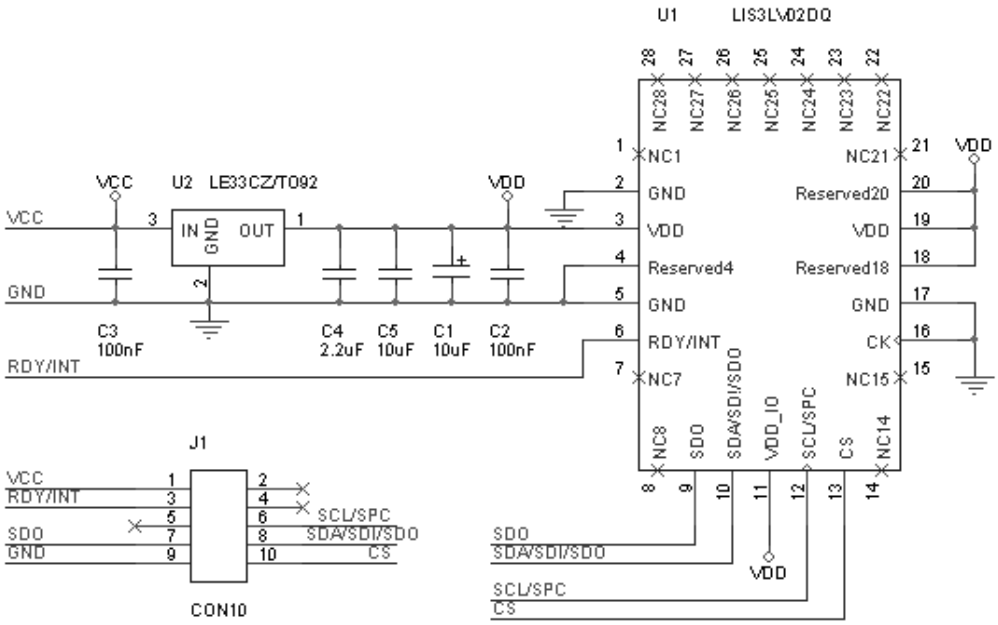
Hlavní funkce akcelometru:

- Volitelný plný rozsah: ±2g nebo ±6g
- Pracuje v napěťovém rozsahu 2,16V až 3,6 V
- I2C/SPI digitální výstupní rozhraní
- Volitelný 12b nebo 16b datová reprezentace
- Přerušení aktivované pohybem nebo programovatelným přerušením
- Vestavený vlastní test, který umožňuje testovat mechanické, a elektronické časti senzoru

Přídavný modul byl navržen a konstruován v předchozí práci, kde lze nalézt postup návrhu a realizace [1]. Na obr. 2.4 je obrázek přídavného modulu a na obr. 2.5 obvodové schéma.



Obrázek 2.4 - Přídavný modul s digitálním akcelometrem LIS3LV02DQ



Obrázek 2.5 - Obvodové schéma přídavného modulů s digitálním akcelometrem LIS3LV02DQ

2.2.2. Přídavný modul s analogovým akcelometrem MMA7260Q

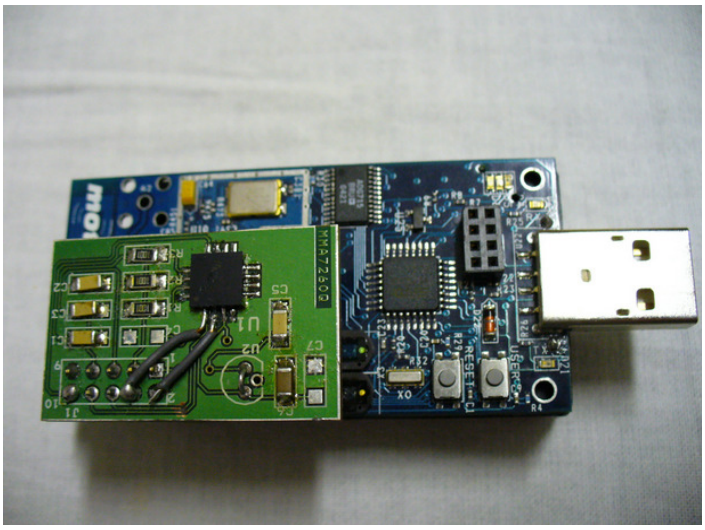
MMA7260Q je nízko nákladový kapacitní Akcelometr. Má 1 pólový filtr s dolní propustí, teplotní vyrovnávaní a g Select, který umožňuje výběr mezi čtyřmi úrovněmi citlivostí. Modul má továrně nastaven nulový g offset v celém rozsahu a mezní frekvenci filtru. Odpadá tak tedy nutnost externího zařízení pro nastavení modulu. Také nabízí spánkový režim, který je pro zařízení napájené pomocí baterii výhodné. [9] Hlavní funkce akcelometru:

- Nastavitelná citlivost ± 1.5g, ±2g, ±4g nebo ±6g
- Nízká spotřeba při měření 500µA
- Spotřeba v režimu spánku 3 µA

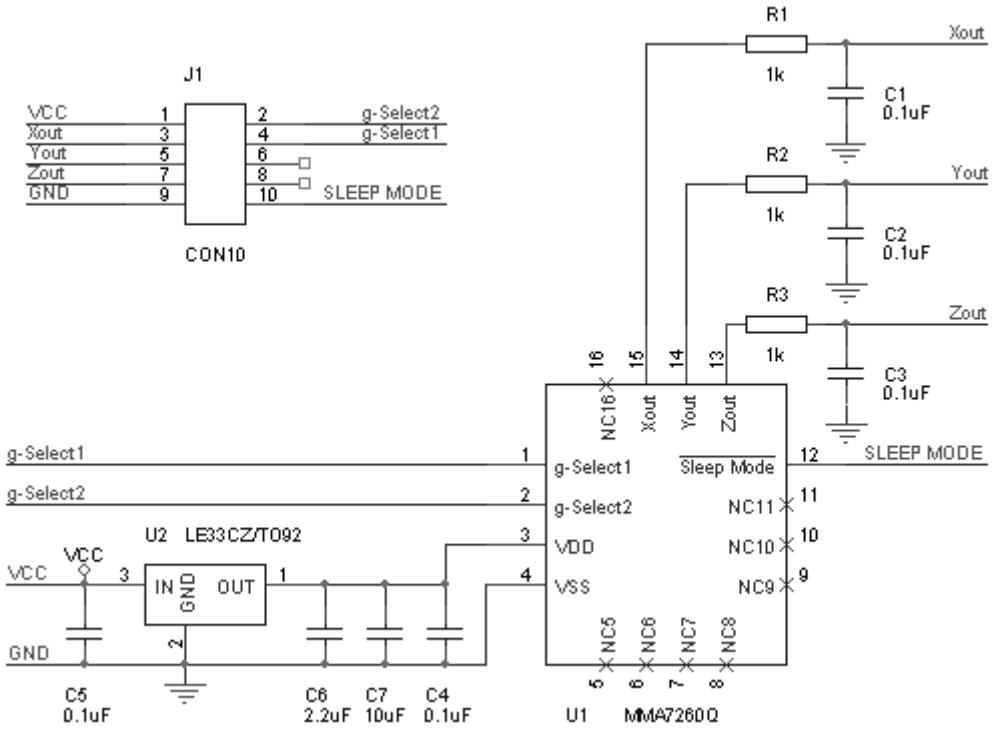
- Nízké napájecí napětí 2,2V – 3,6 V
- Velká citlivost (800 mV/g @1.5 g)
- Rychlý přechod z vypnutého stavu do zapnutého
- Vnitřní vzorkovací frekvence 11kHz
- Zabudované předzpracování signálu s filtrem typu dolní propust
- Teplotní vyrovnávání

Pro komunikaci s platformou TelosB je využíván ADC převodník, který poskytuje přídavný konektor.

Přídavný modul byl navržen a konstruován v předchozí práci, kde lze nalézt postup návrhu a realizace [1]. Na obr. 2.6 je obrázek přídavného modulu a na obr. 2.7 obvodové schéma.



Obrázek 2.6 - Přídavný modul s analogovým akcelometrem MMA7260Q



Obrázek 2.7 - Obvodové schéma přídavného modulů s analogovým akcelometrem MMA7260Q

3. Operační systém TinyOS

3.1. Úvod

TinyOS je bezplatný open-source operační systém, založený na práci s komponenty, zaměřený na bezdrátové senzorové sítě. TinyOS je vestavný operační systém, napsány v programovacím jazyce nesC (nástavba jazyka C) a je tvořen souborem spolupracujících úloh a procesů. Je určen pro zabudování do mále bezdrátové sítě mikroelektronickým systémů senzorů, robotů nebo zařízení vybavených bezdrátovou komunikací, které jsou určeny k detekci světla, vibrací a teploty. Aktuální verze je TinyOS 2.1, která není kompatibilní z předchozí verzí TinyOS 1. x, která byla využitá předchozí prácí. [14]

3.2. Popis TinyOS

Každá hardwarová komponenta má svou softwarovou komponentu, která s ní komunikuje na nejnižší softwarové úrovní ISO/OSI modelu a zprostředkovává vazby mezi vyššími vrstvami. V některých případech je pro daný modul importováno více vrstev, které jsou využívány podle potřeby vývojáře, například rádiový přenos dat. V případě sériové komunikace lze na nejnižší vrstvě přenášet jednotlivé byty, na vyšší vrstvě to je speciální datový rámec a na nejvyšší úrovni probíhá komunikace datových rámců s potvrzováním příjmu. Podle typu aplikace se lze nalinkovat na kteroukoliv vrstvu a ta pak sama komunikuje s vrstvami na nižší úrovni. [10]

Implementace aplikace v TinyOS se opírají o celou řadu komponent, například časovače, arbitrace, bezdrátová nebo sériová komunikace, tlačítka atd. Hlavní myšlenkou komponent je rozdělení kódu, tak abychom zjednodušili strukturu programu a zvýšili univerzálnost těchto aplikací. Jednoduchou změnou v komponentech můžeme změnit bezdrátovou komunikaci na komunikaci přes sériové rozhraní. Jelikož je TinyOS open-source, tak můžeme najít velké množství knihoven a komponent pro různé platformy a příklady implementace senzorů, arbitrace, komunikace na těchto platformách. Komponenty použité v těchto příkladech, lze volně použít ve vlastní aplikaci nebo vytvořit vlastní komponenty, založené na na těchto hotových komponentách.

Aplikace v TinyOS je tvořena na základě souboru události, příkazů a úkolů. Události nastávají, buď po vykonání Split-phase operací nebo po hardwarovém přerušení. Díky tomu se zjednoduší plánování operací v procesoru a zpřehlední program vytvořený uživatelem. Nezkušený uživatel si musí dávat pozor na zpracování události tak, aby nedocházelo k časově neefektivním prodlevám nebo v nejhorším případě k zacyklení programu. Aby se předešlo těmto situacím, využívají se úkoly, které se vykonávají při nevytížení procesů a mají menší prioritu než události. O vykonávání úkolu se v TinyOS stará plánovač úloh. Příkazy se používají k volání funkce z jiného souboru nebo pro zpřehlednění kódu. V TinyOS se využívá Split-phase operace pro příkazy, které mají dlouhou dobu vykonání operace, a mohlo by dojít k zablokování programu. Ve

Split-phase systému, při požadování dlouhého chodu operace, se volání vrátí okamžitě a až po dokončení operace nastane událost tzv. zpětné volání. Tento přístup se nazývá Split-phase, protože rozdělí průběh a dokončení operace do dvou samostatných etap. Split-phase rozhraní umožňují TinyOS komponentům snadno spustit několik činností najednou a nechat provádět paralelně. Také mohou ušetřit paměť. Na obr. 3.1 můžete vidět rozdíl mezi blokovou a Split-phase operací.

Obrázek 3.1 - Porovnání blokové a Split-phase operace

3.3. NesC

NesC je programovací jazyk v operačním systému TinyOS. Jedná se o nástavbu programovacího jazyka C, který používá vlastní překladač NesC. Aplikace v NesC se skládá ze dvou částí: modulu (modules) a konfigurace (configurations). Modul obsahuje kód pro jednotlivé komponenty a konfigurace se užívá pro spojení komponentů do jednoho celku. Moduly implementují rozhraní, které jsou v NesC obousměrné tzn. mohou přijímat nebo zpřístupňovat události a příkazy (funkce). Modul získá požadované rozhraní pomocí konfigurace, která spojuje modul s komponentem, poskytujícím potřebné rozhraní.

3.4. Podpora TinyOS

Praci s TinyOs lze provádět pod Linuxem i Windows. Ve Windows je potřeba nainstalovat linuxový příkazový řádek a příslušný softwarový balíček podporující toto prostředí (například Cygwin). Další možnost je program, který vytvoří virtuální stroj, ve kterém běží Linux s nainstalovaným TinyOS. Ve své práci jsem použil druhou možnost. Podrobný popis lze nalézt na stránkách TinyOS, kde se používá XubunTOS Virtual Machine běžící na WMware serveru [11].

4. Ovládače k přídavným modulům s akcelometry

Ke každému přídavnému modulu s akcelometrem jsem napsal v NesC ovladače, které se dají použit v TinyOS 2.x jako komponenty K práci s nimi stačí jen připojit v konfiguračním souboru tento komponent a v souboru modulu, už můžete, pomocí rozhraní s těmito akcelometry pracovat. Kód ovladačů je napsán tak, aby se dal použít i pro jiné aplikace. Tyto ovladače jsou funkční jen v operačním systému TinyOS 2. x, jelikož kódy v TinyOS 1. x nejsou kompatibilní v TinyOS 2. x. Ovladače pro TinyOS 1. x můžete najít v práci, na kterou navazuji [1].

4.1. Ovládače pro přídavný modul s digitálním akcelometrem

Jak už bylo uvedeno v kapitole 2., tak s přídavným modulem se komunikuje pomocí rozhraní I2C. Na platformě TelosB je sdílena sběrnice, na které pracuje rozhraní I2C s rozhraním radiové komunikace. To je jeden z důvodu, proč se přešlo na novější operační systém TinyOS 2.x. V TinyOs 1.x nebylo řešeno přepínání mezi těmito rozhraními, tak aby mohly pracovat v jedné aplikaci obě dvě. Proto i v předchozí práci [1] tento modul fungoval jen s připojením přes sériové rozhraní a nebyl tedy použitelný pro bezdrátovou senzorovou síť. V TinyOS 2.x se tento problém řeší pomocí arbitrace, kde se tzv. přiděluje právo používat zdroj(např. sběrnici) a podle potřeby se přiděluje právo k přístupu ke zdroji. V mém případě se přiděluje sběrnice rozhraní I2C nebo rozhraní radiové komunikace.

4.1.1. Rezervované zdroje

V TinyOS se rozlišuje mezi třemi druhy rezervovaných zdrojů: vyhrazenými, virtualizovanými a sdílenými. Rezervovaný zdroj je vyhrazený tehdy, když představuje zdroj, ke kterému potřebuje subsystém neomezený exkluzivní přístup a tento zdroj už s nikým dalším nesdílí. Virtualizace rezervovaných zdrojů poskytuje zdroj klientům navzájem skrz softwarovou virtualizaci. Každý klient z virtualizováného rezervovaného zdroje pracuje se zdrojem, jako kdyby ho měl vyhrazený jen pro sebe, ale ve skutečnosti se zdroj mezi klienty přepíná multiplexem. Jelikož virtualizace je udělaná softwarově, tak je omezení počtu klientů dáno jen pamětí nebo výkonnostním omezením. Řízení virtualizace rezervovaných zdrojů je prováděno automaticky a neexistuje žádné rozhraní, které by nabízelo ruční ovládání řízení.

Vyhrazený rezervovaný zdroj je užitečný, když rezervovaný zdroj je vždy řízen jedním komponentem. Virtualizace rezervovaných zdrojů je vhodná, pro klienty, kteří chtějí sdílet zdroj tak, že jím nevadí, rozdělení jejich kontroly mezi všechny klienty na časové úseky.

V některých případech není možné pro klienty používat virtualizaci rezervovaných zdrojů, protože potřebují vykonat řadu operací rychle a za sebou, což

jím virtualizace nedovolí a může dojít k přerušení sledu potřebných operací. Tento problém řeší tzv. sdílení rezervovaných zdrojů. Na platformě TelosB rádiové rozhraní a přídavný modul (I2C rozhraní) sdílí stejnou sběrnici a rádio i přídavný modul musí vykonávat řadu operací za sebou. Kdyby došlo k přerušení sledu operací, tak by zařízení nefungovalo správně. Proto je vhodné pro sdílení této sběrnice využít rezervování zdrojů.

V TinyOS je za přepínání sdíleného zdroje mezi různými klienty zodpovědný arbiter. Určuje, který klient má přístup k rezervovanému zdroji, když se uvolní nebo je zdroj volný. Když klient získá rezervovaný zdroj, tak přebírá celou a neomezenou kontrolu nad tímto zdrojem. Arbiter přepokládá, že klienti mezi sebou spolupracují a měli by při získaní zdroje provést operace, které potřebují. Držení rezervovaného zdroje by mělo být jen v nezbytném čase. Poté by měl klient uvolnit rezervovaný zdroj, protože arbiter nemá možnost odebrat násilně rezervovaný zdroj [20].

4.1.2. Arbiter

Všechny sdílené rezervované zdroje musí mít své arbitrované řízení, které poskytuje klientům daný zdroj. Vzhledem k tomu, že arbiter je v centrálním místě, tak ví, zda je nějaký zdroj momentálně používán. Proto muže poskytovat řadu informací užitečných pro řízení. Arbiter musí poskytovat parametrizací rozhraní rezervovaných zdrojů (Resource) a také informaci o stavu zdroje (ResourceInfo). S rozhraním musí pracovat různí klienti a ti potřebují informaci, jestli je zdroj užíván, k tomu slouží rozhraní ArbiterInfo. Toto rozhrání nám poskytne globální informaci o stavu rezervovaného zdroje (jestli ho někdo používá a kdo ho používá). Arbiter také poskytovatuje parametrizaci rozhraní ResourceRequested a užití parametrizace rozhraní ResourceConfigure. Dále může poskytovat rozhraní ResourceDefaultOwner nebo nějaké jiné, které určuje speciální politiku arbitrace [20].

4.1.3. Práce s rezervovanými zdroji

Klient žádá o rezervovaný zdroj pomocí volání request. Pokud je rezervovaný zdroj volný vrátí se zpět SUCCESS a vyvolá se událost granted. Když je rezervovaný zdroj obsazený, tak se vrátí SUCCESS, ale požadavek bude zařazen do fronty. Když klient nebude, už potřebovat rezervovaná zdroj, tak zavolá release příkaz, ten uvolní rezervovaný zdroj. Dalšímu klientovi, který požádal o zdroj, bude přidělen zdroj a vykoná se u něj událost granted. Když klient žádá víckrát za sebou o přidělení rezervovaného zdroje, vrátí se mu zpět EBUSY a jeho dotaz není přidělen do fronty. Je to ochrana proti monopolizace přidělování zdroje. Další možností o žádání o rezervovaný zdroj je pomocí volání příkazů immediateRequest. Příkaz vrací SUCCESS nebo FAIL. Když vrátí SUCCESS, tak následně je vyvolána událost granted. V případě příchodu FAIL, je zdroj obsazen a požadavek o zdroj není zařazen do fronty. Klient musí zkusit přístup k rezervovanému zdroji později. Klient může ještě využit příkazu isOwner, aby zjistil, jestli je zdroj přiřazen jemu. Všechny tyto příkazy patří do rozhraní Resource.

Rozhraní ArbiterInfo nám dovoluje zjistit, jestli je zdroj užíván, případně kým, pomocí příkazů inUse a clientId. Dalším rozhraním ResourceRequested, zjistíme, jestli jiný klient nežádá o rezervovaný zdroj. Lze to využít například při potřebě využívat rezervovaný zdroj do té doby, než si jiný zdroj o něj požádá. Toto rozhraní poskytuje dvě události requested a immediateRequested, kdy se první vyvolá při volání jiného klienta příkazu request a druhá při volání příkazu immediateRequest. Rozhraní ResourceConfigure umožňuje automaticky konfigurovat rezervovaný zdroj, ještě před přidělením klientovi. Někdy je třeba poskytovaný zdroj konfigurovat pro určité operace, které používá klient. Každý klient má své id a podle něj se pozná, jaká konfigurace je třeba. Rozhraní poskytuje dva příkazy configure a unconfigure, kdy při přidělení rezervovaného zdroje dojde k příkazu configure a při odevzdání zdroje k příkazu unconfigure.

Poslední rozhraní, o kterém se zmíním je ResourceDefaultOwner. Tomuto rozhraní lze přidělit jen jednoho klienta. Klientovi je přiřazen rezervovaný zdroj jen za podmínky, že tento zdroj již nevyužívá jiný klient [12].

4.1.4. Provedení arbitrace

Rozhraní USART, I2C a SPI u platformy TelosB používají stejnou sběrnici. Pro jejich arbitraci použiji rozhraní, které jsem předchozí podkapitole popisoval. Jelikož TinyOS 2. x už nabízí hotové komponenty pro arbitraci této sběrnice [13] a jejich přesný rozbor by byl nad rámec mé práce, tak se v další části, už budu zabývat jen její implementaci.

4.1.5. Konfigurační soubor

První částí ovládače je konfigurační soubor, ve kterém se spojují komponenty nutné pro ovládání přídavného modulu s akcelometrem. Konfigurační soubor:

```
#include "I2C.h"
configuration LIS3LV02DQC{
 provides{
   interface LIS3LV02DQ;
 }
}
implementation {
 components LIS3LV02DQP,new
Msp430I2CC(),MainC,LedsC,HplMsp430GeneralIOC;
  LIS3LV02DQ = LIS3LV02DQP.LIS3LV02DQ;
  LIS3LV02DQP.I2CResource -> Msp430I2CC.Resource;
  LIS3LV02DQP.I2CPacket -> Msp430I2CC.I2CBasicAddr;
  LIS3LV02DQP.Leds -> LedsC;
 LIS3LV02DQP.CSpin -> HplMsp430GeneralIOC.Port63;
 LIS3LV02DQP.Boot -> MainC;
}
```

Z kódu lze vidět, že se implementují komponenty MainC, LedeC, Msp430I2CC, HplMsp430GeneralIOC a LIS3LV02DQP. Komponent MainC je hlavní komponent, který poskytuje bootovaní. LedC dovoluje ovládat Led diody na TelosB, které slouží jako vizuální signalizace. HplMsp430GeneralIOC komponent poskytuje ovládání pinu na přídavném konektoru, který se využije k zapínání zařízení. Nejdůležitější komponent Msp430I2CC mi nabízí rozhraní pro rezervování zdroje a rozhraní I2C. Komponent se stará i o přiřazení id klientovi pro sdílení rezervovaného zdroje. V konfiguračním souboru můžete vidět, že poskytuje rozhraní LIS3LV02DQ. Toto rozhraní mi poskytuje příkazy pro ovládání čtení a zápisu akcelometru. Tyto příkazy jsou napsány v souboru Modul a jejich popis bude uveden později.

4.1.6. Soubor Modul

V Modulu využiji následující rozhraní:

```
uses interface Resource as I2CResource;
uses interface I2CPacket<TI2CBasicAddr>;
uses interface Boot;
uses interface Leds;
uses interface HplMsp430GeneralIO as CSpin;
```

Rozhraní Boot mi poskytuje událost booted, která nastává prvním spuštěním a slouží k prvnímu nastavení (v mém případě zapnutí akcelometru).

```
event void Boot.booted() {
  call CSpin.makeOutput();// It sets hight pin 10 and turn on accelometer.
  call CSpin.set();
}
```

Pro zapnutí akcelometru se přivádí logická jednička na pin 10. K tomu potřebuji rozhraní HplMsp430GeneralIO(v kódu přejmenované na CSpin), které mi poskytne ovládání tohoto pinu. Pomoci příkazu makeOutput nastavím pin jako výstup a příkazem set nastavím logickou jedničku. Těmito příkazy jsem zapnul akcelometr.

Další důležitá částí kódu jsou příkazy, které poskytuji v rozhraní a pomocí kterých se dá nastavovat nebo číst z akcelometru.

```
command error_t LIS3LV02DQ.writeNReg(uint8_t* p_data, uint8_t
num_data) {
  atomic{
      status = WRITE_REGISTER;  // writting to the register
      i=0;
     for(;i<3;i++){
        data_write[i] = p_data[i];
     }
```

```
    }
    return call I2CResource.request();
}

command error_t LIS3LV02DQ.readNReg(uint8_t* start_reg, uint8_t
num_data) {
    atomic{
      status = WRITE_FROM_READING;//writting register address
      nd=num_data;
      start_register_autoinc=start_reg;
    }
    return call I2CResource.request();
}
```

Jelikož příkazy jsou Split-phase, tak se musí v kódu nastavit default události, které se překryjí v Modulu, z kterého se volají dané příkazy.

```
default event error_t LIS3LV02DQ.writeNRegDone() {
  }
default event bool LIS3LV02DQ.readNRegDone(uint8_t* pointer_data, uint8_t
num_data) {
    return SUCCESS;
  }
```

V příkazu se užívá další rozhraní a to Resource, pomocí kterého se žádá o přiřazení sběrnice, na které funguje I2C. Příkaz writeNReg slouží k nastavení akcelometru. Příkaz požaduje při svém volání dva parametry adresu pole dat a počet dat. Pole dat obsahuje adresu prvního registru, který chceme přepsat a následně nové hodnoty registrů. Tyto hodnoty registrů jsou řazený v poli dat za sebou, tak jak jdou adresy těchto registrů za sebou (například adresa registru 1, hodnota registru 1, hodna registru 2,…). Příkaz readNReg se používá ke čtení dat z akcelometru. Zde příkaz potřebuje jeden parametr a to adresu, ze které chceme číst data z akcelometru(v mém případě data os).

Jelikož je volán rezervovaný zdroj, tak v kódu musí být implementována událost granted, která se vykoná při získání zdroje.

```
event void I2CResource.granted(){
    // writing to the register
    if(status==WRITE_REGISTER){
      call I2CPacket.write(I2C_START | I2C_STOP, I2C_slaveAdrr ,nd, da-
ta_write)
    }
    //writing register address for reading
    if(status==WRITE_FROM_READING){
      call I2CPacket.write(I2C_START|I2C_STOP, I2C_slaveAdrr,
1,&start_register_autoinc)
   }
}
```

Zde už jde vidět, že se využívá rozhraní pro I2C. Jelikož komunikace v I2C funguje jako master – slave, tak musíme při čtení nebo zápisu vždy dát prvně příkaz jednotce slave(akcelometru) co má udělat. Při zápisu do registru po prvním odeslaném datovém rámci (adresy kam), začne akcelometr přijímat data. Při požadavku pro čtení, po prvním odeslaném datovém rámci, začne odesílat data z dané adresy.

Příkaz je Split-phase a proto nám nastane událost writeDone. V této události musíme rozlišovat mezi zápisem do registru nebo jeho čtením.

```
async event void I2CPacket.writeDone(error_t error, uint16_t addr, uint8_t
length, uint8_t* data){
   if(error==SUCCESS){
     if(status==WRITE_FROM_READING){
          status==READING;
              if( call I2CPacket.read(I2C_START\I2C_STOP, I2C_slaveAdrr,
number,
             data_read)==SUCCESS){
              }
     }
    if(status==WRITE_REGISTER){
     signal LIS3LV02DQ.writeNRegDone();
     call I2CResource.release();
    }
}
```

Při zápisu už nepotřebujeme další prácí s I2C a proto následuje příkaz release, který odevzdá sběrnici a vyvolání události writeNRegDone. U čtení zavoláme příkaz read a přečteme data z akcelometru. Následně se nám vyvolá událost

4.2. Ovládače pro přídavný modul s analogovým akcelometrem

Modul komunikuje s platformou TelosB pomoci ADC převodníku, který nabízí přídavný konektor. Proto tento modul nabízí jednoduchou komunikaci, protože na rozdíl od I2C komunikace se nemusí dělit s rádiovou komunikací o sběrnici, tím si ušetří práce s arbitrací. Nastavování modulu je taky jednoduché, protože se nedělá pomocí zápisu do registru, jako u předchozího modulu, ale jen přepínáním logické úrovně na nastavovacích pinech.

4.2.1. ADC převodník

Analogově-číslicové převodníky (ADC) jsou zařízení, které přeměňují analogový vstup na digitální výstup, klasicky z napětí na binární hodnotu. TinyOS 2.x narozdíl od starší verze oddělil do dvou nezávislých komponentu nastavení ADC zařízení a jeho řízení [12]. Jelikož budu pracovat se třemi vstupy do ADC převodníku, tak se musí udělat jejich nastavení (konfigurace). TinyOs 2.x obsahuje komponent hamamatsu S10871(fotodioda) v opt/tinyos-2.x/tos/platforms/telosa/chips/s10871, který využívá stejný ADC převodník. Proto jsem využil tento komponent a udělal jsem v něm úpravy v nastavení použitého kanálu.

```
msp430adc12_channel_config_t config = {
   inch: INPUT_CHANNEL_A0,               // input channel
   sref: REFERENCE_VREFplus_AVss,      // reference voltage
   ref2_5v: REFVOLT_LEVEL_1_5,         // reference voltage level
   adc12ssel: SHT_SOURCE_ACLK,        // clock source sample-hold-time
   adc12div: SHT_CLOCK_DIV_1,         // clock divider sample-hold-time
   sht: SAMPLE_HOLD_4_CYCLES,       // sample-hold-time
   sampcon_ssel: SAMPCON_SOURCE_SMCLK,   // clock source sampcon
signal
   sampcon_id: SAMPCON_CLOCK_DIV_1         // clock divider sampcon
signal
  };
```

Jediné v čem se liší nastavení, je vstupní kanál, který je v mém případě A0, A1, A2 (osa x, y a z). Další část je řídící, která díky rozdělení na konfiguraci a řízení je nezávislá na zařízení a proto můžu využít hotové rozhraní Read, kde se při spojení zadává jen velikost hodnoty, kterou má vracet. Proto jsem vytvořil tři nové komponenty, které slouží k nastavení třech vstupů do ADC převodníku a poskytují rozhraní pro jejich čtení.

4.2.2. Vytvořeni konfiguračního souboru

Nyní když jsem udělal komponenty pro ADC kanály, můžu udělat komponent pro ovládání přídavného modulu analogového akcelometru. Znovu začnu

popisem konfiguračního souboru, který propojuje rozhraní, které budu potřebovat.

```
configuration MMA7260QC {
 provides{
   interface MMA7260Q;
 }
 }
implementation
{
 components MMA7260QP;
 components MainC;
 components LedsC;
 components new Adc0pC() as A0; // control adc
 components new Adc1pC() as A1; // control adc
 components new Adc2pC() as A2; // control adc
 components HplMsp430GeneralIOC;  // control pin on mode
 MMA7260QP.Boot -> MainC;
 MMA7260QP.Leds -> LedsC;
 MMA7260Q=MMA7260QP.MMA7260Q;
 MMA7260QP.Read0 -> A0; // cteni z adc A0
 MMA7260QP.Read1 -> A1; // cteni z adc A1
 MMA7260QP.Read2 -> A2; // cteni z adc A2
 MMA7260QP.CSpin -> HplMsp430GeneralIOC.Port63; // sleep mode pin
 MMA7260QP.CSping1 -> HplMsp430GeneralIOC.Port34; //select g1 pin
 MMA7260QP.CSping2 -> HplMsp430GeneralIOC.Port35; //select g2 pin
}
```

V konfiguračním souboru implementuji komponenty MainC, LedC, HplMsp430GeneralOC, Adc0pC, Adc1pC a Adc2pC. Komponent MainC je hlavní komponent, který poskytuje bootovaní. LedC dovoluje ovládat Led diody na TelosB, které slouží jako vizuální signalizace. Pomocí HplMsp430GeneralIOC komponentu získám ovládání pinů na přídavném konektoru. Jeden pro zapínání sleep módu (pin 10), a pak další dva piny (pin 6 a 7) na nastavování citlivosti akcelometru. Poslední tři komponenty Adc0pC, Adc1pC a Adc2pC jsou komponenty, které jsem připravil pro kanály ADC převodníku, pomoci kterých čtu hodnoty os x, y a z. Dále poskytuje rozhraní s příkazy pro nastavování a čtení z akcelometru.

4.2.3. Vytvoření souboru Modul

V modulu využijeme následující rozhraní.

```
  interface Read<uint16_t> as Read0; // reading axis X
  interface Read<uint16_t> as Read1; // reading axis Y
  interface Read<uint16_t> as Read2; // reading axis Z
  interface Boot;
  interface Leds;
  interface HplMsp430GeneralIO as CSpin;   //sleep mode
```

```
  interface HplMsp430GeneralIO as CSping1; // select g1
  interface HplMsp430GeneralIO as CSping2; // select g2
```

Rozhraní Boot poskytuje událost booted, která nastává prvním spuštěním. Slouží k prvnímu nastavení, v tomto případě zapnutí a nastavení citlivosti akcelometru.

```
event void Boot.booted() {
  /* -------------------- Default g-select ----------- */
  call CSpin.makeOutput();/* Sleep mode OFF*/
  call CSpin.set();
  call CSping1.makeOutput();/* Select g1 - low*/
  call CSping1.clr();
  call CSping2.makeOutput();/* Select g2 - low*/
  call CSping2.clr();
 }
```

Pro zapnutí akcelometru potřebuji vypnout sleep mód. K tomu potřebuji rozhraní HplMsp430GeneralIO (v kódu přejmenované na CSpin), které mi poskytne ovládání daného pinu. Pomocí příkazu makeOutput nastavím pin jako výstup a příkazem set nastavím logickou jedničku. Dále se nastavuje základní citlivosti, kde jde na pin g1 a g2 logická nula (clr). V tabulce 4.1 můžete vidět možná nastavení citlivosti.

g-Select2	g-Select1	g-Range	Sensitivity
0	0	1.5g	800mV/g
0	1	2g	600mV/g
1	0	4g	300mV/g
1	1	6g	200mV/g

Tabulka 4.1 - Možná nastavení citlivosti akcelometru

Další důležitá část kódu jsou příkazy, které poskytuji v rozhraní a pomocích kterých se dá nastavovat nebo číst z akcelometru. Začnu popisem příkazů writeNReg, které poskytují nastavení citlivosti akcelometru a sleep mód.

```
command error_t MMA7260Q.writeNReg(uint8_t gSelect) {
  /*--------------------select g------------------*/
  if(gSelect==0){
      call CSpin.makeOutput();         //Sleep mode OFF
        call CSpin.set();
        call CSping1.makeOutput(); // Select pin - g1
        call CSping1.clr();
        call CSping2.makeOutput(); // Select pin – g2
        call CSping2.clr();
  }
 …
 …
  if(gSelect==3){
```

```
        call CSpin.makeOutput();
            call CSpin.set();
            call CSping1.makeOutput();
            call CSping1.set();
            call CSping2.makeOutput();
            call CSping2.set();
     }

    /*-----------------Sleep mode-----------------*/
    if(gSelect==4){
        call CSpin.makeOutput();
            call CSpin.clr();
            call CSping1.makeOutput();
            call CSping1.clr();
            call CSping2.makeOutput();
            call CSping2.clr();
    }
    signal MMA7260Q.writeNRegDone();
    return SUCCESS;
}
```

Volání příkazu writeNReg nabízí čtyři možnosti nastavení, pomocí poslání parametru hodnoty 0 – 4. Hodnota 0 až 3 slouží k nastavení parametru citlivosti akcelometru, která odpovídá tabulce 4.1. Hodnota 4 je pro uspání akcelometru. Ze spánku akcelometr dostaneme voláním příkazu writeNReg s hodnotou 0 až 3. Z kódu lze vidět využití rozhraní HplMsp430GeneralIO k ovládání pinu.

Poslední příkaz, který poskytuje čtení dat z akcelometru je readNReg.

```
command error_t MMA7260Q.readNReg() {
    call Read0.read();
    return SUCCESS;
}
```

Jak je vidět tak příkaz využívá rozhraní Adc0pC, které poskytuje příkaz read. Tento příkaz je Split-phase a proto po přečtení následuje událost readDone.

```
event void Read0.readDone(error_t result, uint16_t data)
  {
    if (result == SUCCESS){
      read_data[0]= data;
      call Read1.read();
    }
  }
```

Zde je vidět, že se provede uložení hodnoty a volání čtení pomocí rozhraní Adc1pC. Volaný příkaz je zase Split-phase a vyvolává po přečtení readDone. Kód této události vypadá podobně, jen s rozdílem, že se ukládá hodnota do read_data [1] a volá se příkaz čtení z rozhraní Adc2pC, který také vyvolá událost readDone.

```
event void Read2.readDone(error_t result, uint16_t data)
 {
  if (result == SUCCESS){
    read_data[2]= data;
    signal MMA7260Q.readNRegDone(read_data);// send date back
  }
 }
```

Zde dojde k uložení hodnoty a následně odeslání hodnot z akcelometru pomocí vyvolání události readNRegDone.

Jako v předchozím ovladači nesmí chybět default události k příkazům.

```
default event error_t MMA7260Q.writeNRegDone() {
 return SUCCESS;
}
default event bool MMA7260Q.readNRegDone(uint16_t* pointer_data) {
   return SUCCESS;
}
```

Ovladač pro přídavný modul s analogovým akcelometrem, už je nyní připraven použití. Stačí použit tento ovladač jako komponent v dané aplikaci.

4.3. Experimenty k ověření funkčnosti ovladačů

K ověření funkčnosti ovladačů pro přídavné moduly s akcelometrem, jsem vytvořil jednoduchou aplikaci, která v nastaveném intervalu posílá do počítače přečtené data z akcelometru. Pro poznání, že akcelometr funguje správně, jsem údaje pozoroval ve třech polohách akcelometru. Jelikož akcelometr udává zrychlení, tak jsem mohl využít zemskou přitažlivost, kdy na zařízení v klidu působí zrychlení g směrem k středu Země. Každá poloha byla vybrána tak, aby směr zrychlení g působil ve směru určené osy.

Na obr. 4.1, obr. 4.2 a obr. 4.3jsou zobrazeny data z digitálního akcelometru. Porovnáním dat při odlišné poloze akcelometru, lze vidět změny hodnoty zrychlení os. Každá osa je na obrázku oddělena čárou a řazení je x, y, z. Na obrázku jsou na jednom řádku zobrazené dva vzorky naměřených dat.

Na obr. 4.4, obr. 4.5 a obr. 4.6 jsou zobrazeny data z analogové akcelometru. Podobně jak u digitálního akcelometru, lze ze vzorku vidět změny hodnot zrychlení os, podle natočeni akcelometru. U analogového akcelometru jsem zvolil směr působení g proti směru osy, abych ukázal, že nulová hodnota zrychlení není v nule.

Z výsledku experimentu vyplývá, že ovladače a akcelometry fungují správně.

62 01	04 00	01 00	62 01	04 00	03 00
62 01	05 00	02 00	62 01	04 00	02 00
62 01	04 00	01 00	62 01	05 00	01 00
62 01	04 00	02 00	62 01	05 00	02 00
62 01	04 00	02 00	62 01	04 00	01 00
62 01	04 00	02 00	62 01	04 00	01 00
62 01	04 00	01 00	62 01	04 00	01 00
62 01	05 00	03 00	61 01	05 00	03 00
62 01	05 00	01 00	62 01	04 00	01 00
62 01	04 00	02 00	62 01	04 00	02 00
61 01	05 00	01 00	62 01	04 00	01 00
62 01	02 00	02 00	62 01	04 00	02 00

Obrázek 4.1 – Natočení digitálního akcelometru tak, aby zrychlení g působilo ve směru osy x

03 00	5A 01	08 00	03 00	5A 01	08 00
03 00	5A 01	08 00	02 00	5A 01	08 00
02 00	5A 01	08 00	02 00	5A 01	08 00
03 00	5A 01	08 00	03 00	5A 01	08 00
03 00	5A 01	08 00	03 00	5A 01	08 00
03 00	5A 01	09 00	03 00	5A 01	08 00
03 00	5A 01	08 00	03 00	5A 01	08 00
03 00	5A 01	08 00	02 00	5A 01	08 00
03 00	5A 01	08 00	03 00	5A 01	08 00
03 00	5A 01	08 00	03 00	5A 01	09 00
03 00	5A 01	08 00	02 00	5A 01	08 00
02 00	5A 01	08 00	02 00	5A 01	08 00

Obrázek 4.2 – Natočení digitálního akcelometru tak, aby zrychlení g působilo ve směru osy y

05 00	0D 00	5C 01	02 00	0D 00	5E 01
00 00	0D 00	5F 01	02 00	0D 00	5D 01
05 00	08 00	5B 01	04 00	0E 00	5E 01
01 00	0E 00	5F 01	01 00	0C 00	5B 01
03 00	08 00	5B 01	03 00	08 00	5B 01
02 00	0B 00	5D 01	02 00	0D 00	5D 01
03 00	10 00	5B 01	03 00	0D 00	5D 01
03 00	0A 00	5E 01	02 00	0D 00	5C 01
04 00	0D 00	5D 01	03 00	0B 00	5E 01
00 00	08 00	5D 01	00 00	0D 00	5C 01
06 00	0F 00	59 01	04 00	0E 00	5D 01
04 00	0D 00	5E 01	01 00	0D 00	5E 01

Obrázek 4.3 – Natočení digitálního akcelometru tak, aby zrychlení g působilo ve směru osy z

05 C5	0E 6A	0E 20	05 CD	0E 5F	0E 3A
05 D5	0E 62	0E 22	05 D9	0E 67	0E 1E
05 E5	0E 67	0E 3F	05 D5	0E 7A	0E 3D
05 B2	0E 4E	0E 0C	05 DB	0E 69	0E 36
05 FC	0E 40	0E 42	05 DD	0E 54	0E 3E
05 E1	0E 63	0E 28	05 E2	0E 7E	0E 15
05 D9	0E 62	0E 32	05 E6	0E 58	0E 2D
05 F8	0E 6C	0E 3A	05 CC	0E 6A	0E 1D
05 C9	0E 69	0E 37	05 E9	0E 7C	0E 2D
05 D0	0E 50	0E 1D	05 C5	0E 53	0E 31
05 CD	0E 76	0E 06	05 B8	0E 63	0E 2B
05 EC	0E 70	0E 25	05 D3	0E 76	0E 2C

Obrázek 4.4 – Natočení analogového akcelometru tak, aby zrychlení g působilo ve směru osy x

0E 1C	06 DD	0F 0E	0E 0B	06 EA	0E FA
0E 40	06 CE	0E F2	0E 1D	06 F1	0E E9
0E 34	06 F9	0E CB	0E 19	06 F9	0E ED
0E 33	06 F8	0E FA	0E 13	06 FC	0E ED
0E 1C	06 FF	0E F9	0D EA	06 FC	0E E4
0E 32	07 02	0E E1	0E 12	06 FE	0E CF
0E 1F	06 EB	0E EC	0E 23	06 ED	0E CC
0E 26	06 E4	0E D9	0E 24	07 10	0E D3
0D F5	06 DA	0E EA	0D F2	06 EA	0E E6
0E 23	06 F3	0E C3	0D FC	07 05	0E DC
0E 13	07 02	0E F5	0E 10	07 15	0E FC
0E 25	06 F2	0E F9	0E 2E	06 DE	0E D9

Obrázek 4.5 – Natočení analogového akcelometru tak, aby zrychlení g působilo ve směru osy y

0E 41	0E 53	07 E2	0E 40	0E 47	07 D3
0E 3D	0E 31	07 C7	0E 3E	0E 4D	07 D5
0E 3C	0E 48	07 EE	0E 48	0E 3D	07 D1
0E 34	0E 3D	07 E6	0E 2F	0E 41	07 E3
0E 40	0E 55	07 D7	0E 59	0E 4F	07 EA
0E 3B	0E 2D	07 F5	0E 43	0E 40	07 CF
0E 65	0E 49	07 F4	0E 1A	0E 48	07 E5
0E 34	0E 4B	07 E6	0E 39	0E 41	07 F0
0E 4A	0E 48	07 D6	0E 37	0E 44	07 E0
0E 41	0E 1E	07 F7	0E 24	0E 3D	07 DC
0E 45	0E 3E	07 F3	0E 07	0E 41	07 E7
0E 23	0E 46	07 ED	0E 43	0E 47	07 E2

Obrázek 4.6 – Natočení analogového akcelometru tak, aby zrychlení g působilo ve směru osy z

5. Návrh komunikačního protokolu

Pro komunikaci mezi zařízením, které měří pohyby pacienta a zařízením, které zpracovává tyto data, je kriticky problém s rychlostí odeslání všech naměřených dat. Měřicí zařízení musí nejméně v každých 20 milisekundách udělat nové měření (50hz). V bezdrátové síti je nejméně 5 měřicích zařízení. Tyto podmínky vytváří časové kritérium, které musím splnit při vytváření komunikačního protokolu. Datová výměna nesmí být moc dlouhá, protože může dojít k přetečení zásobníku v měřících zařízení a měření by pak bylo zmařeno. K tomu může docházet, když v každé smyčce příjímání naměřených dat bude docházet k neustálému narůstání počtu nových naměřených dat. Při použití komunikační protokol Open-ZB Stach v minulé práci [1], se nezdařilo toto časové kritérium splnit, a proto v mé práci navrhuji nový komunikační protokol, který bude splňovat dané kritérium.

5.1. Požadavky na komunikační protokol

Aby mohl celý systém dobře fungovat, je nutné mezi jednotlivými zařízeními správně navrhnout komunikační protokol a vypořádat se co nejlépe s chybami, které mohou nastat. Hlavním úkolem této bezdrátové senzorové sítě je monitorovat pacienta pomocí senzorů umístěných na jeho těle. V síti je centrální jednotka, která dále data zpracovává, a měřicí jednotky, které zaznamenávají pohyb pacienta. Komunikace v této sítí probíhá vždy jen mezi centrální jednotkou a měřicí jednotkou. Komunikační protokol musí řešit problém, který by mohl nastat při vysílání dat ze dvou nebo více měřicích jednotek do jedné centrální jednotky současně, což by způsobilo ztrátu dat nebo znemožnění komunikace. Elegantním řešením je zvolení komunikace mezi centrální jednotkou a měřicí jednotkou Master – Slave. Tato komunikace funguje tak, že Master(centrální jednotka) je jediný, kdo rozhoduje o vysílání a Slave (měřicí jednotka) vždy jen poslouchá do té doby, než je oslovena Masterem. Tím nemůže dojít k tomu, aby vysílaly dvě jednotky Slave najednou.

Komunikační protokol musí v sobě zahrnovat dva základní stavy: inicializaci a měření. Stav inicializace musí zajišťovat připojení měřících jednotek k centrální jednotce a jejich nastavení pro stav měření. Měřicí stav musí zajišťovat přenos naměřených dat a kontrolu stavu měřících jednotek (nastavení, připojení,sleep mode).

Další část komunikačního protokolu je vstup a výstup z vnějšku. V mém případě je to PC, kterému jsou posílány data zpracovaná centrální jednotkou a je mu také umožněno nastavování zařízení.

Jak už bylo zmíněno výše, tak musí být délka zprávy co nejlépe využita. Zpráva by tedy neměla obsahovat zbytečné informace, které by zvětšovaly její délku. Dále by komunikační protokol měl obsahovat možnost komprese, která by podstatně zmenšila počet nutných zpráv a tím dovolila rychlejší komunikaci.

Protože pracuji se zařízením, které je napájeno z baterií, tak komunikační protokol by měl řešit i úsporné režimy, a znovunalezení přístroje při jeho resetu, které například mohlo nastat výměnou baterie.

5.2. Tvorba komunikačního protokolu

5.2.1. Jednotka master – inicializační část

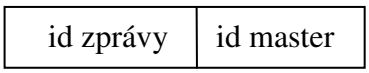
Inicializace v jednotce master nastává při zapnutí nebo resetu přístroje. V této části musí dojít k vytvoření id seznamu jednotek slave, který slouží k adresování zpráv správné jednotce slave. Dále se musí postarat o odeslání nastavující zprávy, která nastaví jednotku slave do požadované konfigurace. Sled kroku, ke kterému dochází po spuštění stavu inicializace, můžete vidět na obr. 5.3.

První krok je odeslání inicializační zprávy, která je adresovaná pro všechny, co poslouchají. Zprávu můžou přijmout jen jednotky slave, které jsou v režimu inicializace. V dalším kroku jednotka master čeká na zprávu od jednotky slave po určitou dobu. Když v této době dojde zpráva od jednotky slave, tak se následně zkontroluje, jestli už je její id přidáno v seznamu id jednotek slave, jestliže není, tak je přidáno. Poté je poslaná nastavovací zpráva dané jednotce slave a přechází se na začátek režimu inicializace. V případě kdy nedojde v určený čas zpráva od jednotky slave, tak se navýší počet nepřijatých odpovědí na inicializační zprávu V případě překročení určeného maxima počtu nepřijatých odpovědí, přechází jednotka master ze stavu inicializace do měření, v opačném případě se vrací na začátek inicializace a znovu odesílá inicializační zprávu.

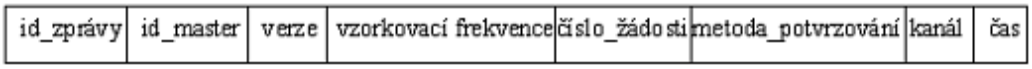
Při odesílání inicializační zprávy může dojít k tomu, že přijde odpověď od více než jedné jednotky slave. Přijata bude jen zpráva od jedné jednotky slave a ostatní zprávy jsou ztraceny. V tomto případě nám nevadí, že zprávy jsou ztraceny, jelikož jednotka slave od které nebyla přijata odpověď, pošle znova odpověď při přijetí další inicializační zprávy. Tímto způsobem budou postupně inicializovaný všechny jednotky slave.

K inicializaci dochází na základním vysílacím frekvenčním kanálu (26), zatímco měření může být prováděno na jiném kanálu.

Složení inicializační zprávy naleznete na obr. 5.1, nastavovací zprávy na obr. 5.2. Zprávy od jednotky slave naleznete v další podkapitole.



Obrázek 5.1 – Inicializační zpráva

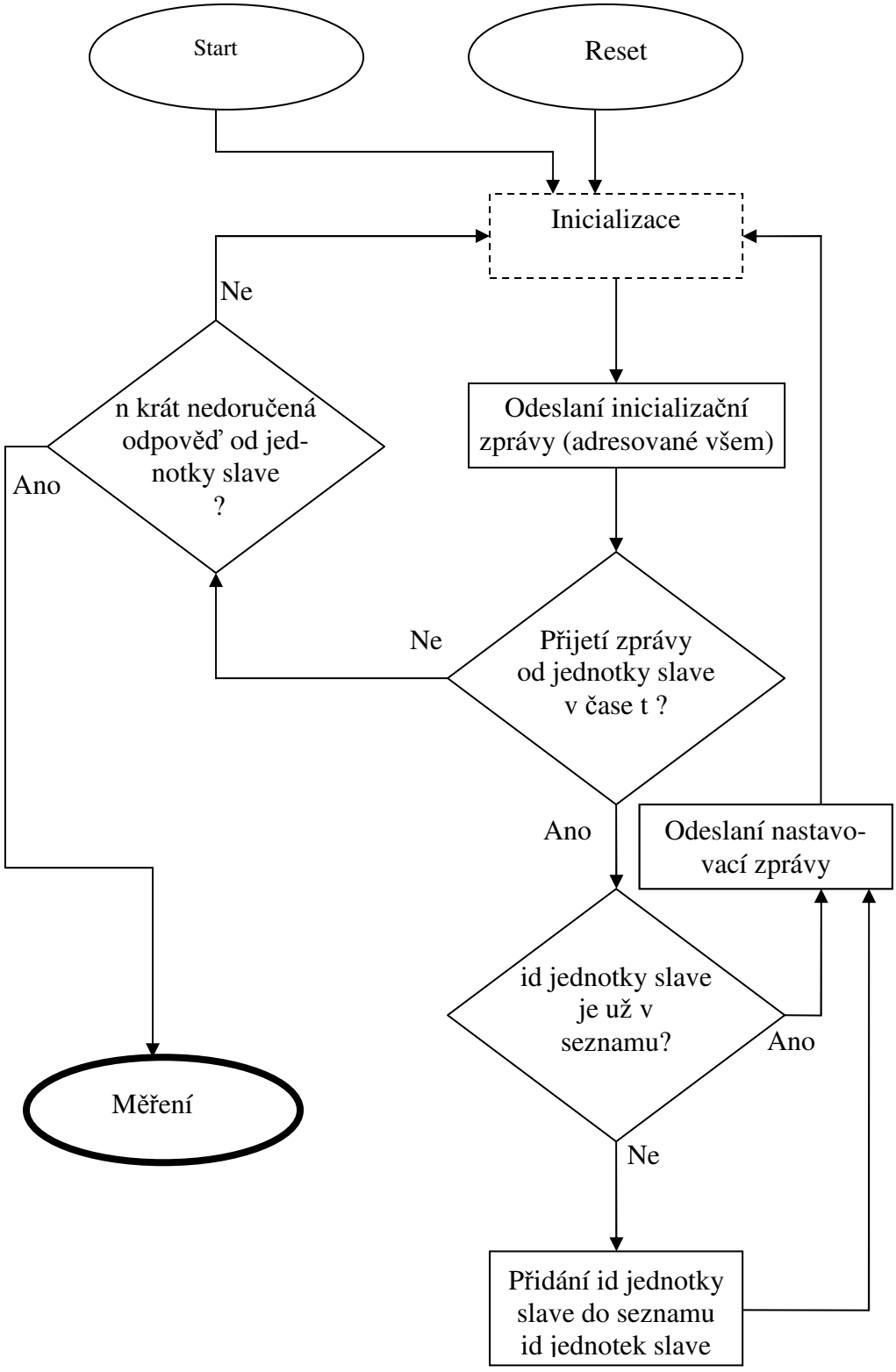


Obrázek 5.2 - Nastavovací zpráva

Inicializační zpráva obsahuje jen id zprávy, pro identifikování typu zprávy a id jednotky master, aby jednotka slave věděla adresu jednotky master.

Nastavovací zpráva už je obsáhlejší, obsahuje navíc verzi nastavovací zprávy, vzorkovací frekvenci, číslo žádosti o měřená data, metodu potvrzování přija-

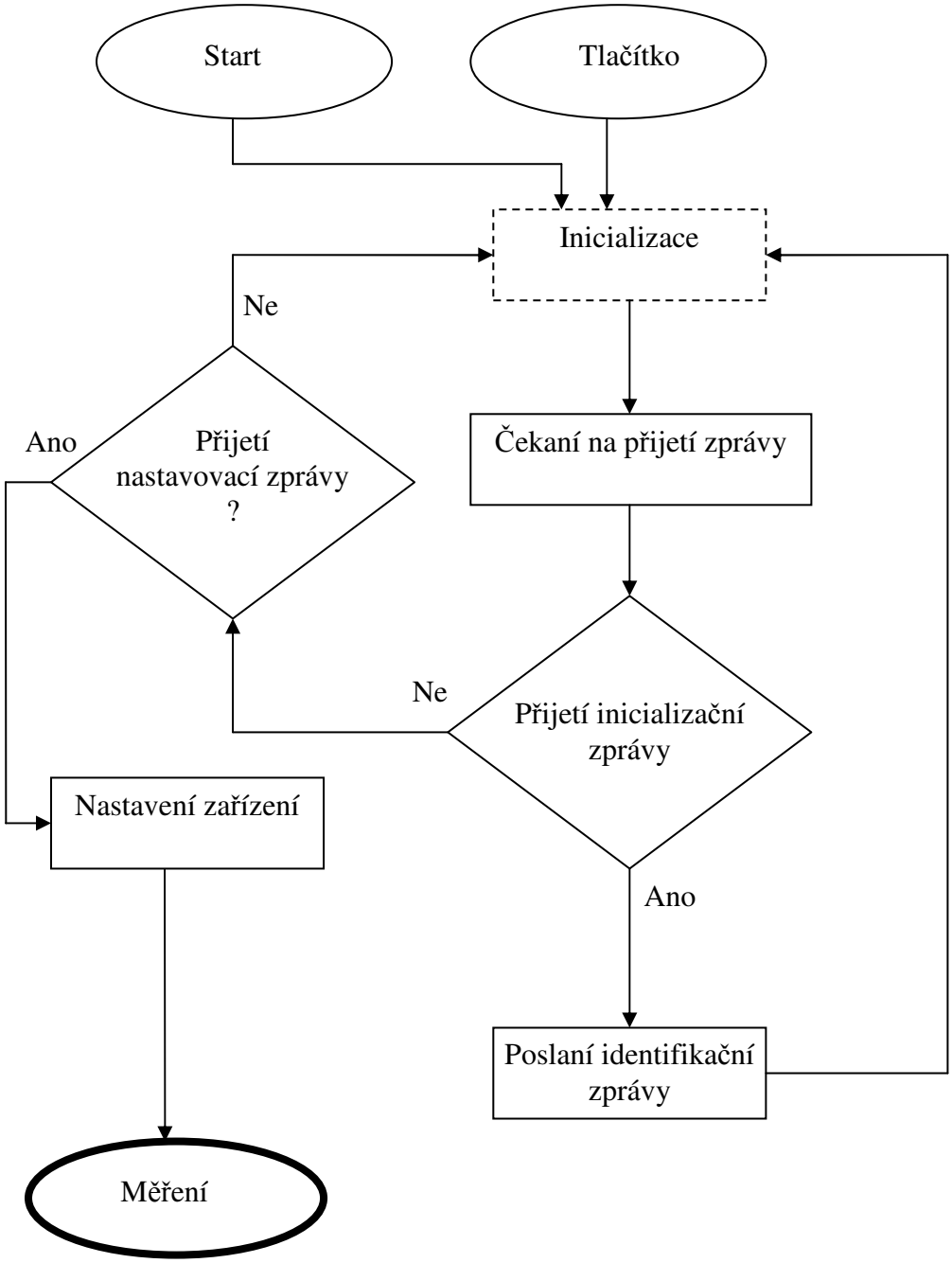
tých dat, vysílací kanál a lokální čas v jednotce master. Při inicializaci je verze a číslo žádosti o měřená data 0. Přesnější použití této zprávy, bude popsaná později.



Obrázek 5.3 – Diagram režimu inicializace u jednotky master

5.2.2. Slave - inicializační část

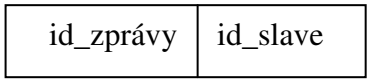
Inicializace u jednotky slave nastává při zapnutí, resetu nebo stisknutí tlačítka na přístroji. Jak už jsem psal výše, tak jednotka slave vždy čeká na dotaz od jednotky master. Proto se nic nevykoná, dokud nepřijde inicializační zpráva nebo nastavovací zpráva od jednotky master. Při přijmutí zprávy od jednotky master se testuje, jestli je zpráva inicializační nebo nastavovací. Jiné zprávy jsou v tomto režimu ignorovaný. Sled kroku, ke kterému dochází po spuštění stavu inicializace, můžete vidět na obr. 5.4.



Obrázek 5.4 – Diagram režimu inicializace u jednotky slave

Při přijmutí inicializační zprávy odpoví jednotka slave identifikační zprávou (obr. 5.5), na kterou jí jednotka master pošle zpět nastavující zprávu. Při přijmutí nastavující zprávy se provede nastavení zařízení a přechod do stavu měření.

Na obr. 5.2 je vidět složení nastavovací zprávy. Pomocí jí se nastaví vzorkovací frekvence pro akcelometr, kanál na kterém se bude vysílat, způsob potvrzovaní posílaných zpráv a rozdíl mezi lokálním časem jednotky master a jednotky slave.



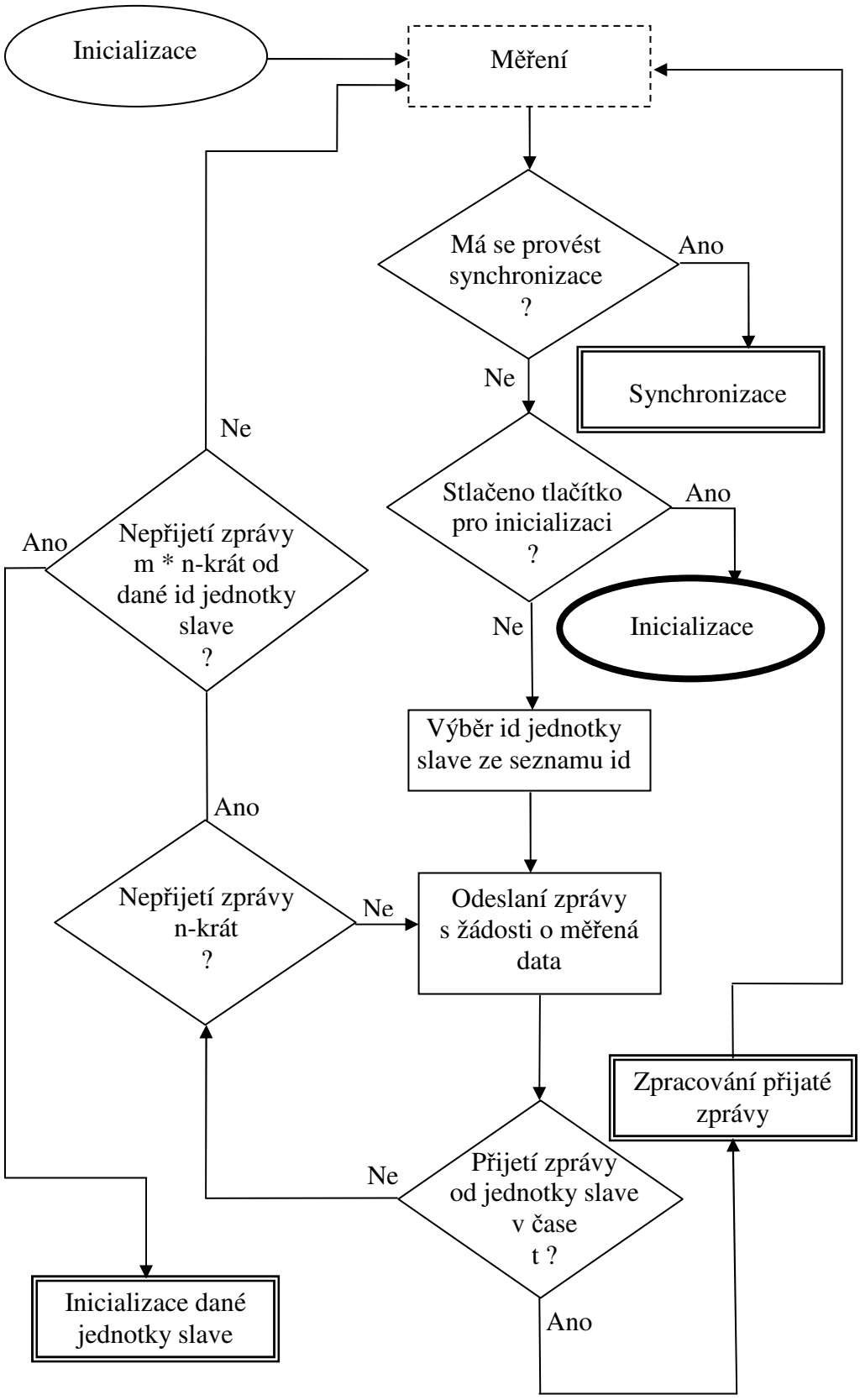
Obrázek 5.5 – identifikační zpráva

5.2.3. Master – měřící část

Do stavu měření se přechází ze stavu inicializace, ze kterého je předán seznam id jednotek slave. V prvním kroku se provádí kontrola, jestli se má provést synchronizace (více v kapitole 5.2.13.) nebo přechod do stavu inicializace po stlačení tlačítka. Jestliže se nemá provést ani jedna z výše uvedených operací, tak se pokračuje výběrem id jednotky ze seznamu id. Vybrané jednotce slave se pošle žádost o naměřená data, poté se čeká po určitý časový úsek na odpověď od tázané jednotky slave.

Při přijetí odpovědi, je tato zpráva zpracována (zpracování zprávy bude popsáno později) a poté se vrací jednotka master zpět do počátečního stavu režimu měření a vybírá se následující jednotka slave ze seznamu. Jestliže v daném časovém úseku nepřijde odpověď od jednotky slave, navýší se hodnota nepřijatých odpovědí na žádost o naměřená data v čítači. Žádost o naměřená data se znova zopakuje, když bude hodnota nepřijatých odpovědi menší než dané maximum. V opačném případě se navýší hodnota nepřijatých odpovědí k tázané jednotce slave v čítači. Poté se přejde do počátečního stavu režimu měření a vybere se ze seznamu následující jednotka slave, za podmínky, že nebylo překročeno maximum nepřijatých odpovědi k dané jednotce slave. Při nesplněni podmínky se přechází do inicializace jednotky slave (více v kapitole 5.2.12.), se kterou nebylo navázáno spojení. Po ukončení inicializace se přechází na začátek režimu měření. Sled popsaných kroků je na obr. 5.6.

Na obr. 5.7 je složení zprávy žádosti o naměřená data, která se skládá z id zprávy, id jednotky master, aktuální verze nastavovací zprávy a číslo žádosti. Zpráva obsahuje číslo aktuální verze nastavovací zprávy, aby jednotka slave mohla udělat kontrolu, že má aktuální nastavení. Když jednotka master projede celý seznam id jednotek slave, tak přechází na začátek seznamu a navýší číslo žádosti. Podle hodnoty čísla žádosti jednotka slave určí, jestli má poslat nové naměřená data nebo zopakovat odeslaní naměřených dat.



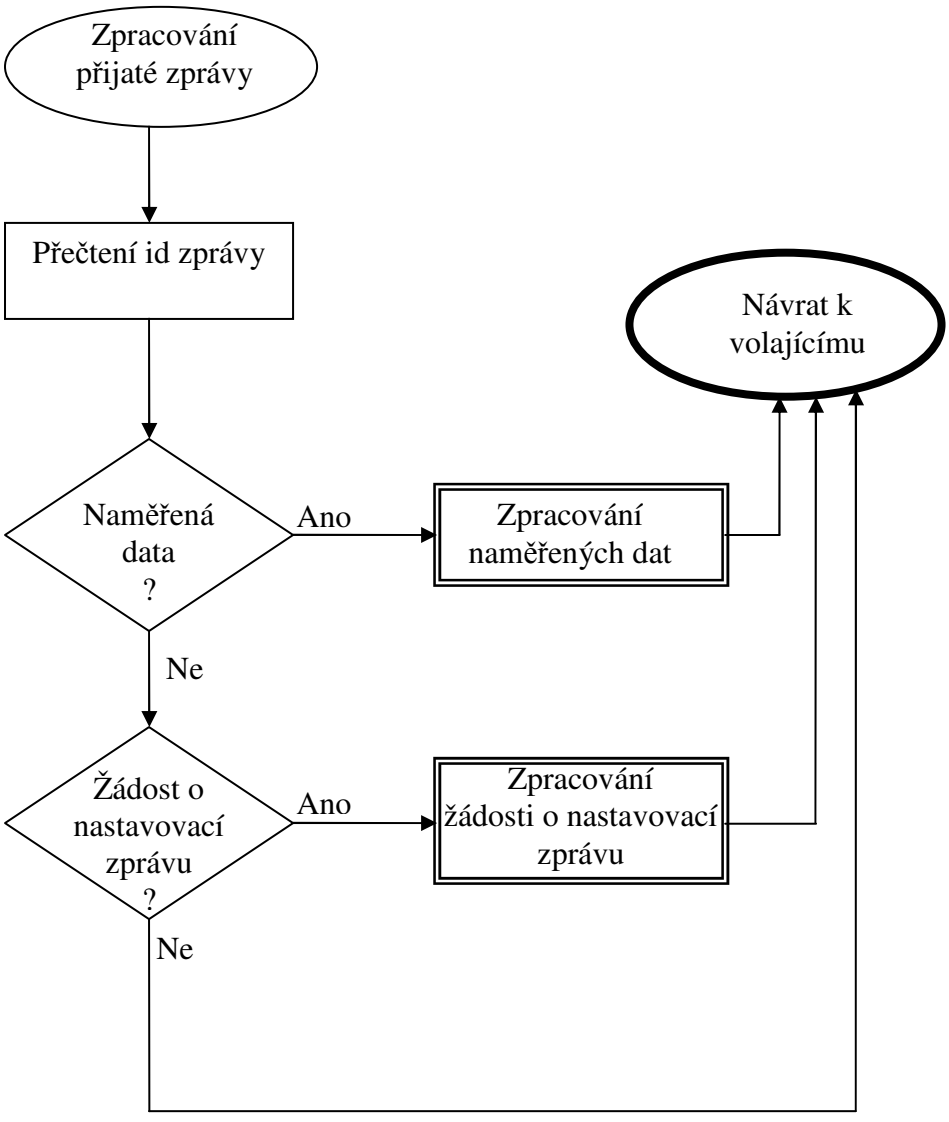
Obrázek 5.6 – Diagram režimu měření v jednotce master

id zprávy	id master	verze	číslo žádosti

Obrázek 5.7 - Zpráva žádost o naměřená data

5.2.4. Master - zpracování přijaté zprávy

Na obr. 5.6 lze vidět, že po přijetí odpovědi od jednotky slave, dochází k zpracování přijaté zprávy. V této podkapitole rozeberu podrobněji toto zpracování zprávy. Na obr. 5.8je diagram zpracování přijaté zprávy, ze kterého vycházím.

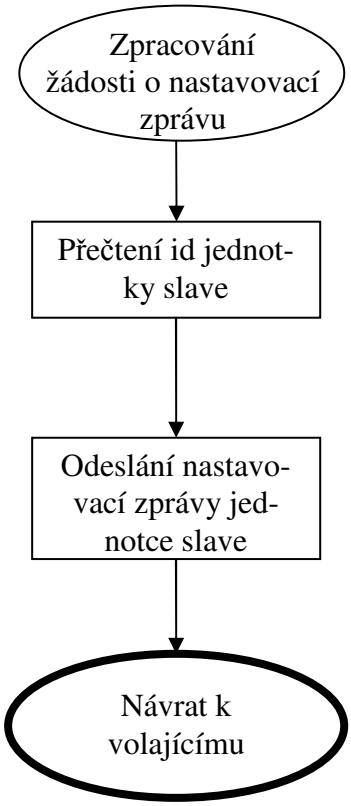


Obrázek 5.8 Diagram zpracování zprávy jednotkou master

Každá zpráva obsahuje id zprávy, podle které rozeznám typ zprávy a podle toho s ní pak mohu naložit. V tomto režimu mohou přijít jen dva druhy zpráv, které se mohou zpracovávat, ostatní jsou ignorovaný. Jsou to zprávy s naměřenými daty nebo žádost o nastavovací zprávu.

5.2.5. Master - zpracování žádosti o nastavovací zprávu

Žádost o nastavovací zprávu přijde od jednotky slave, když zjistí, že nemá aktuální verzi nastavení ze zprávy žádost o nová naměřená data. Ve zprávě je udáno id jednotky slave, na které se odešle nastavovací zpráva. Na obr. 5.9 je zobrazen postup zpracování.



Obrázek 5.9 – Diagram zpracování žádosti o nastavovací zprávu

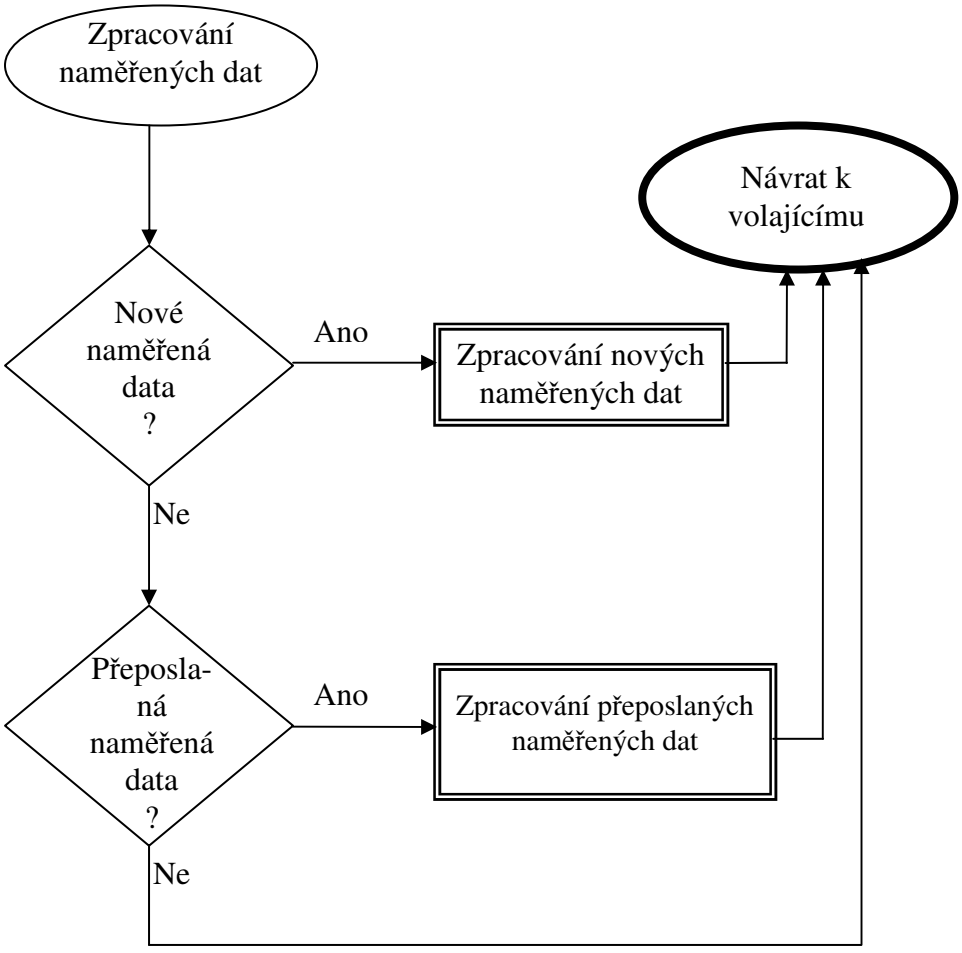
Na obr. 5.10 je zobrazeno složení zprávy žádost o nastavovací zprávu. Skládá se z id jednotky slave a verze, která je v jednotce slave.

id_zprávy	id_slave	verze

Obrázek 5.10 - Zpráva žádost o nastavovací zprávu

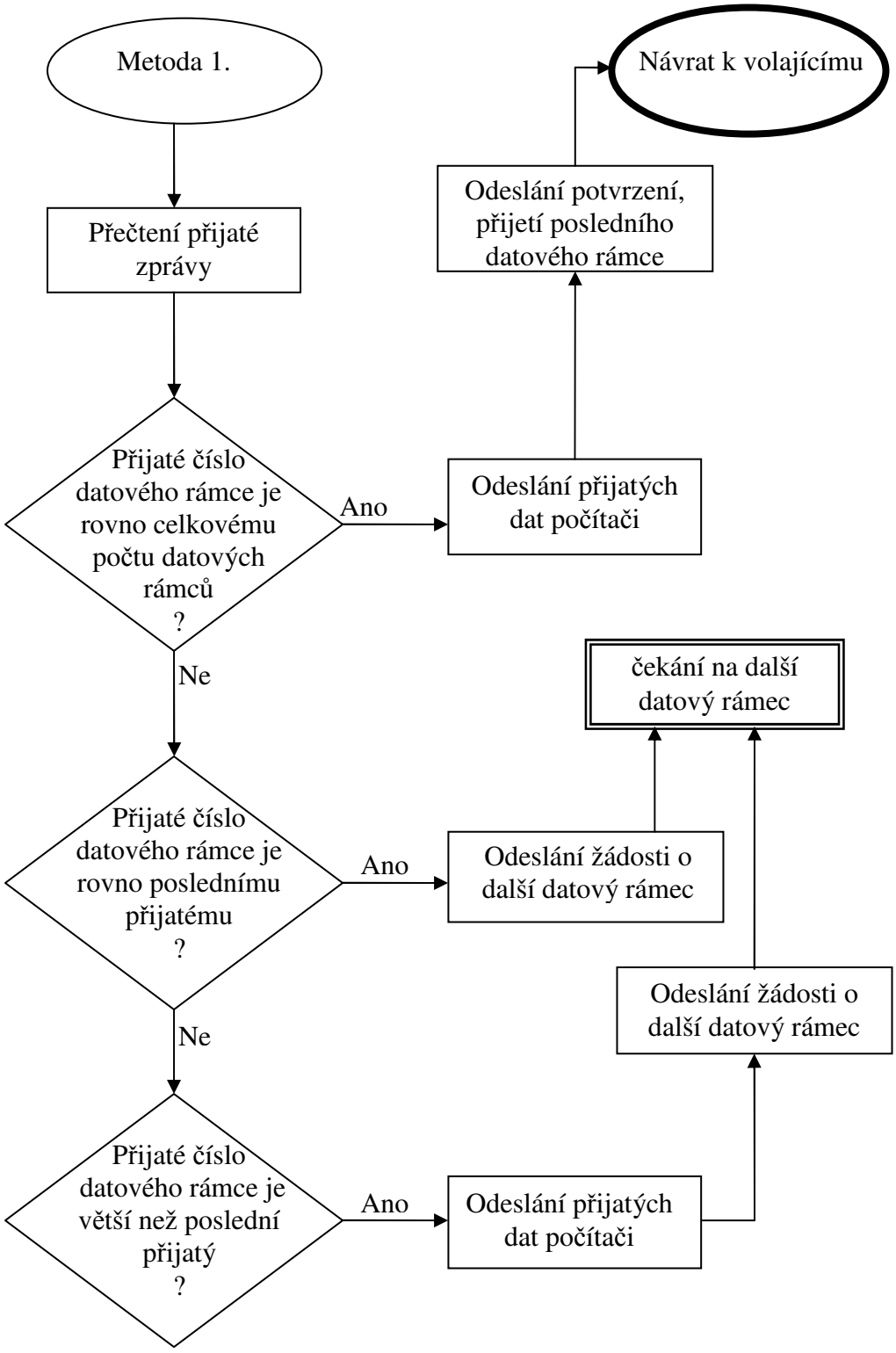
5.2.6. Master – zpracování naměřených dat

Rozlišují se tři druhy zpráv, které obsahují naměřená data. Jedna z nich je zpráva, která obsahuje nově naměřená data bez komprese, další je zpráva, která obsahuje nově naměřená data s kompresí a poslední zpráva obsahující přeposlaná data, o které si jednotka master požádala. S novými naměřenými daty se postupuje jinak než s přeposlanými daty, jak je zobrazeno na obr. 5.11. Při posílaní naměřených dat jsou tři způsoby potvrzování jejich přijetí. První způsob je, že se potvrzuje přijetí každého datového rámce. Další způsob potvrzuje jen přijetí všech datových rámců, a když chybí jeden nebo více datových rámců, tak se pošle žádost o opětovné zaslání všech datových rámců s naměřenými daty. Poslední způsob potvrzování je kombinace obou předchozích způsobů. Kontroluje se přijetí každého datového rámce, ale posílá se potvrzení, až po přijetí všech datových rámců od jednotky slave. Potvrzení obsahuje čísla datových rámců, které nebyly doručeny. V případě, že byly doručeny všechny, tak se odešle potvrzení o bezchybném přijetí všech datových rámců. Každý typ potvrzování dále určuje způsob zpracování příchozích datových rámců s naměřenými daty.



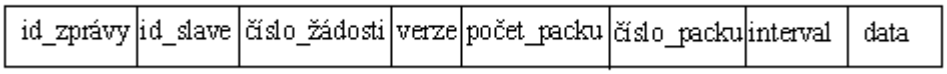
Obrázek 5.11 – Diagram zpracování naměřených dat

5.2.7. Master - zpracování naměřených dat pomocí prvního typu potvrzování



Obrázek 5.12 – Diagram zpracování nových naměřených dat bez komprese 1. metodou potvrzování

Obr. 5.12 ukazuje, jak pracuje první možnost potvrzování pro režim bez komprimace. Zpráva s naměřenými daty bez komprimace obr. 5.13 obsahuje id jednotky slave, číslo dotazu od jednotky master, verzi, celkový počet datových rámců, číslo datového rámce, vzorkovací frekvenci akcelometru a data (dva vzorky měření s časem).

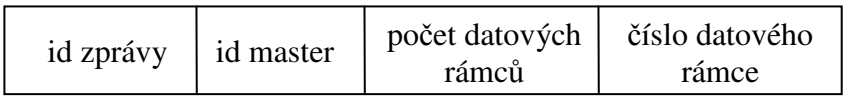


Obrázek 5.13- Nekomprimovaná zpráva s naměřenými daty

Důležité hodnoty pro tento typ potvrzování jsou číslo datového rámce a celkový počet datových rámců. Z diagramu na obr. 5.12 vidíte, že se v prvním kroku porovnává, jestli přijaté číslo datového rámce není i poslední (pomocí celkového počtu datových rámců). Shoduji-li se tyto hodnoty, tak jednotka master odešle potvrzení o přijetí všech datových rámců. V případě že se nerovnají, tak se porovnává přijaté číslo datového rámce s posledním přijatým, které je v případě, že ještě žádný datový rámec nepřišel rovno nule. Rovnají-li se hodnoty, tak to znamená, že nedošla jednotce slave žádost o další datový rámec (potvrzení o přijetí datového rámce), proto se znova tato zpráva odešle. V případě že přijaté číslo datového rámce je vetší než poslední přijatý datový rámec, tak přijatý datový rámec je ten správný a dojde k odeslání dat počítači. Následně se jednotce slave zašle žádost o další datový rámec.

Tento typ potvrzování zaručuje přijetí všech datových rámců. Jednotka slave nepošle další datový rámec, pokud jednotka master nepošle žádost o další datový rámec. Nedojde-li jednotce master datový rámec, tak jednotka master zopakuje žádost o zaslání dalšího datového rámce, která není žádostí o další datový rámec, ale o nedoručený datový rámec. Kdyby se nepodařilo i po několika žádostech o další datový rámec, přijmout datový rámec od jednotky slave, tak se to považuje za nedoručení odpovědí od dané jednotky slave. Tato situace byla popsána v podkapitole 5.2.3., kdy dojde buď k přechodu k další jednotce slave nebo k přechodu do inicializace.

Potvrzovací zpráva je zobrazena na obr. 5.14.



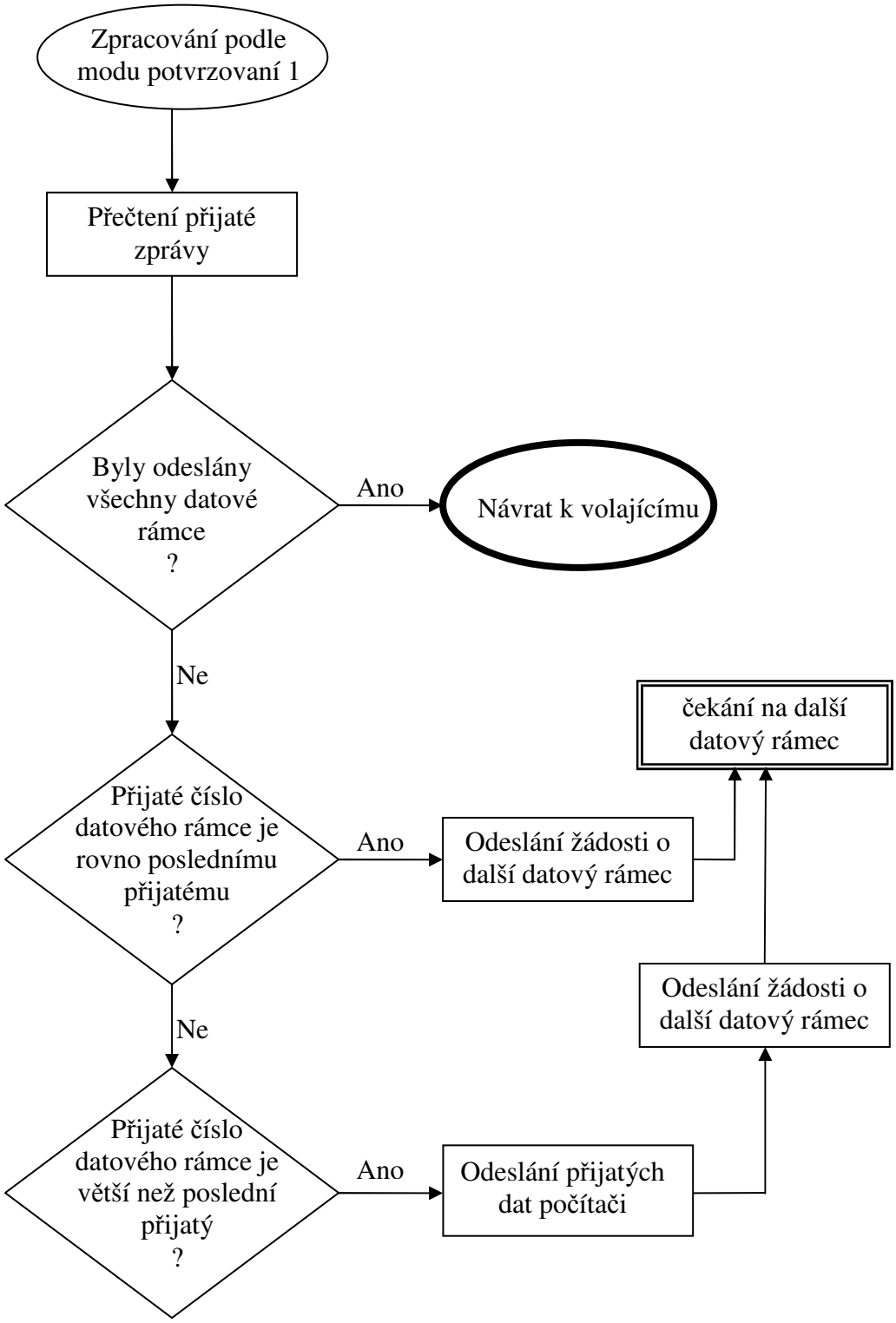
Obrázek 5.14 – Potvrzovací zpráva

U komprimované zprávy není předem znám počet datových rámců, proto jsem u tohoto typu potvrzování přidal novou zprávu, která přijde při odeslání všech dat. Nahrazuje porovnávání, jestli poslední číslo datového rámce se rovná celkovému počtu datových rámců, ostatní kroky jsou stejné a jsou zobrazeny na obr. 5.16.

Složení zprávy oznamující, že byly odeslány všechny datové rámce, je na obr. 5.15. Číslo měření bude vysvětleno později.

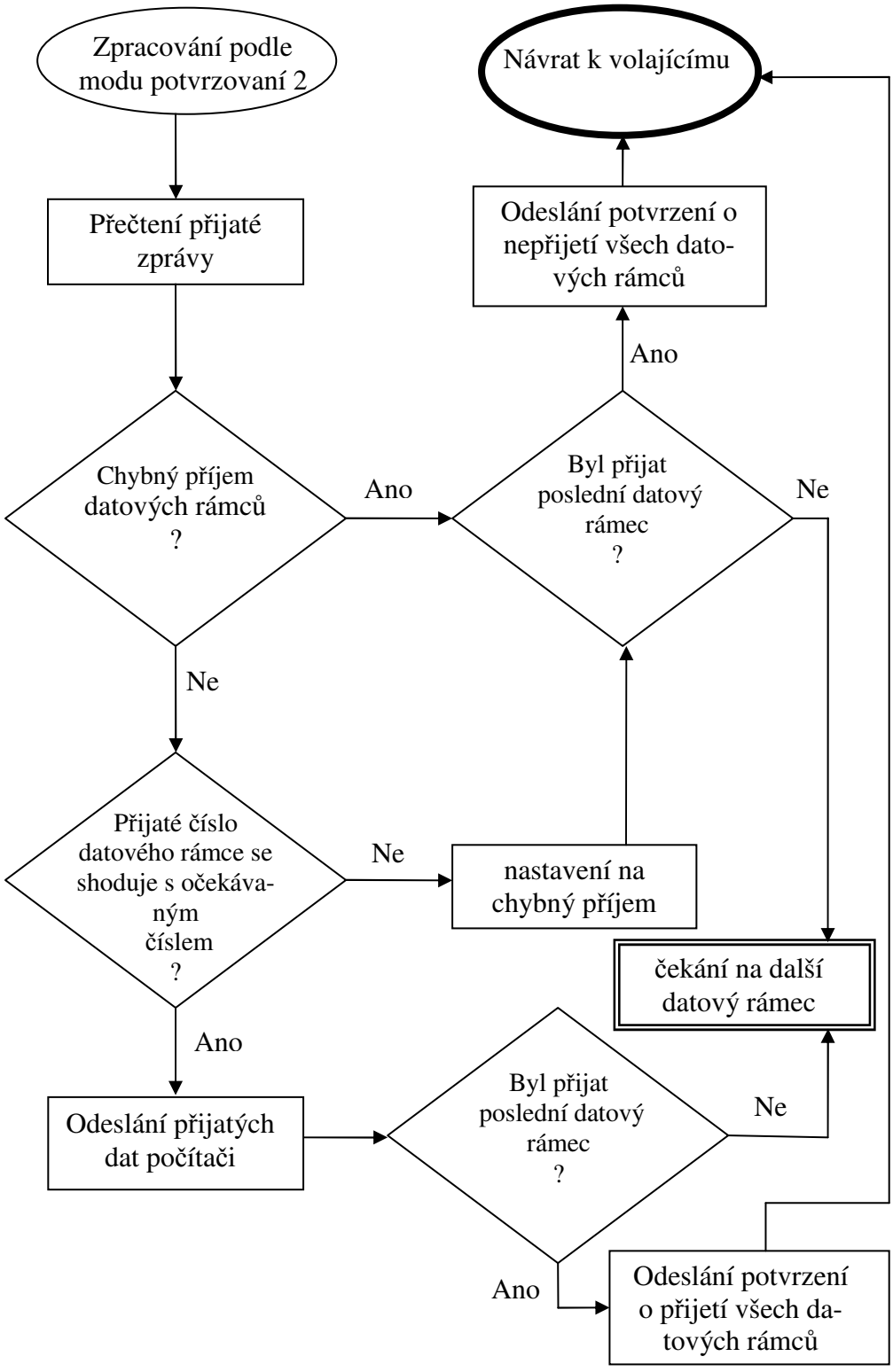
id zprávy	id slave	počet datových rámců	číslo měření

Obrázek 5.15 – Zpráva o odeslání všech datových rámců



Obrázek 5.16 - Diagram zpracování nových naměřených dat s kompresí 1. metodou potvrzování

5.2.8. Master - zpracování naměřených dat pomocí druhého typu potvrzování



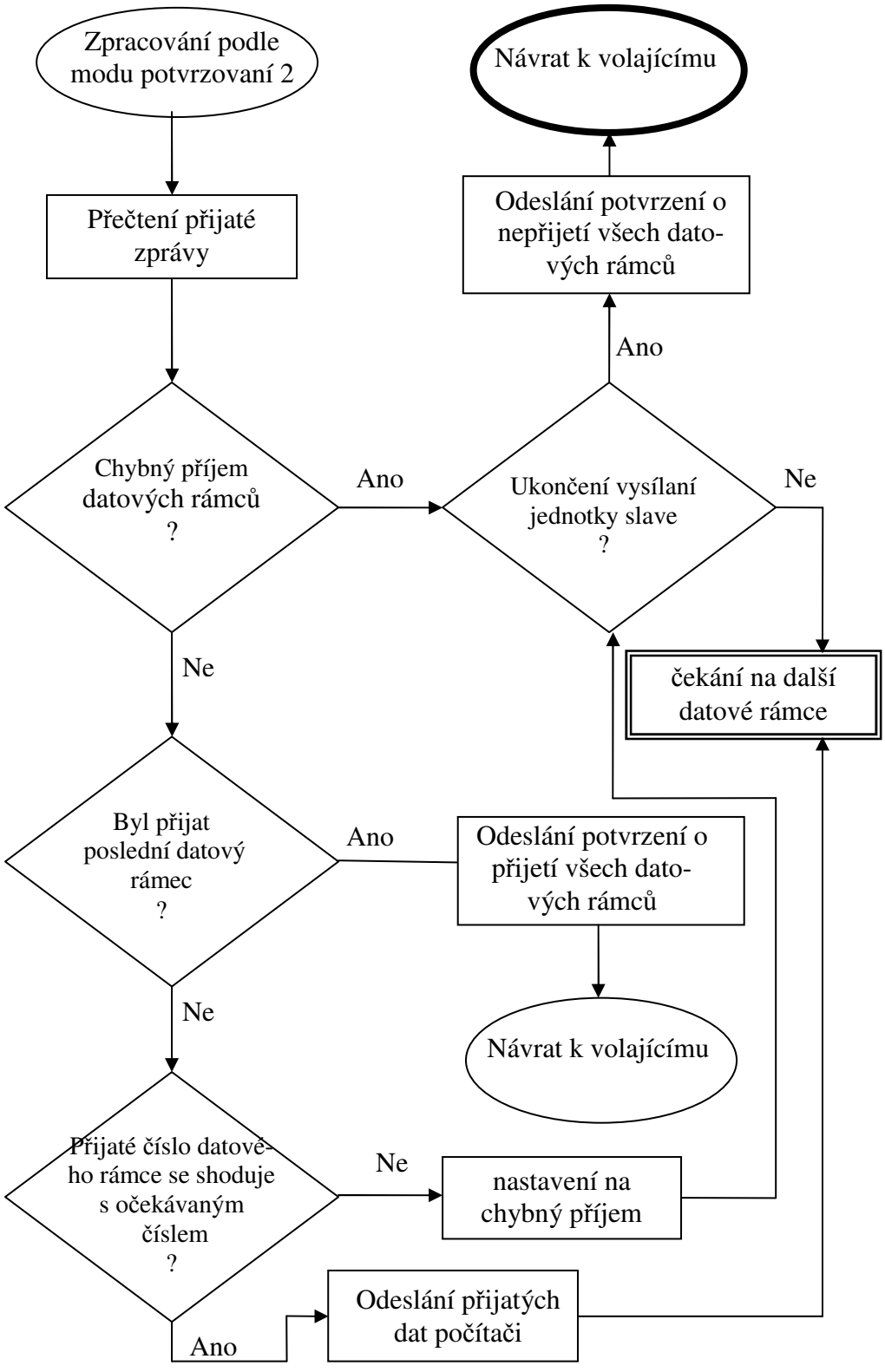
Obrázek 5.17 - Diagram zpracování nových naměřených dat bez komprese 2. metodou potvrzování

Tento typ potvrzování jsem navrhl pro komunikaci, kde je minimální ztráta dat. Zmenšením kontrolního mechanizmu z každého datového rámce na celkový balík datových rámců, se docílí větší rychlosti komunikace master – slave. Algoritmus posílání dat v tomto režimu potvrzování je následující. Jednotka master pošle žádost o nová naměřená data a jednotka slave odpoví posláním zpět všech datových rámců s daty a následně čeká na potvrzení o doručení všech dat. V případě posílání komprimovaných dat, se pošle jako poslední datový rámec zpráva s počtem odeslaných datových rámců obr. 5.15.

Zpracování přijatého datového rámce bez komprimace je na obr. 5.17. Jednotka slave odesílá datové rámce za sebou, a proto se provádí kontrola, jestli přicházejí datové rámce za sebou. Na obr. 5.17 vidíte, že se nejprve kontroluje, jestli v předchozím příjmu datového rámce nedošlo chybě. Dojde-li k chybě, tak už se neprovádí kontrola, zda jdou příchozí datové rámce za sebou, ale jen jestli už přišel poslední datový rámec, což je konec vysílání jednotky slave. Po přijmutí posledního datového rámce, se odešle zpráva o chybném přijetí datových rámců a ukončí se potvrzovací režim. Když daný datový rámec není poslední, tak se přijímají dále datové rámce od jednotky slave, dokud nedojde k přijetí posledního datového rámce. Nedojde-li v předchozí kontrole chybě, tak následuje kontrola, jestli přijatý datový rámec je následující minulého datového rámce. Jestliže není, tak se nastaví chyba kontroly a provede se kontrola, jestli přijatý datový rámec byl poslední. Při přijmutí posledního datového rámce se odešle zpráva o chybném přijetí datových rámců, v opačném případě se přijímají další datové rámce od jednotky slave, dokud nepřijde poslední datový rámec, až pak se odešle zpráva o chybném přijetí datových rámců. Po odeslání této zprávy se ukončí potvrzovací režim. Jestliže přijímaný datový rámec je následující minulého datového rámce, tak se pošlou data počítači a následně se zkontroluje, jestli je datový rámec poslední. V případě že je, tak se odešle potvrzovací zpráva o přijetí všech datových rámců a ukončí se potvrzovací režim, jinak pokračuje kontrola dál.

Na obr. 5.18 je potvrzovací režim pro komprimovaná data. U komprimace neznáme konečný počet datových rámců na začátku, proto je na konec odesílání přidaná zpráva o konci odesílání naměřených dat obr. 5.15. U zpracování zpráv s kompresí se v prvním kroku kontroluje, jestli byl obdržen poslední datový rámec . Jedná-li se o poslední datový rámec, tak dojde k odeslání potvrzení o přijetí všech datových rámců v pořádku. Nejedna-li se o poslední datový rámec, tak následuje kontrola, jestli přišel následující datový rámec. Když přijde správný datový rámec, tak se odešlou data do počítače a čeká se na další datový rámec. Kdyby nedošel správný datový rámec, tak následuje stejný postup, jak na obr. 5.17.

Potvrzovací zpráva, že došly všechny datové rámce a zpráva o chybě při přijmu datových rámců, má stejné složení (obr. 5.14). Rozdíl jednotka slave rozezná pomocí čísla a celkového počtu datových rámců, kdy při doručení všech datových rámců se číslo datových rámců shoduje s celkovým počtem datových rámců.



Obrázek 5.18 - Diagram zpracování nových naměřených dat s kompresí 2. metodou potvrzování

5.2.9. Master - zpracování naměřených dat pomocí třetího typu potvrzování

Tento typ potvrzování je kombinací dvou přecházejících způsobů potvrzování. Neposílá potvrzení po každém přijatém datovém rámci, ale po dokončení odesílaní všech datových rámců od jednotky slave. Jednotka master odešle jednotce slave jen čísla datových rámců, které nebyly doručeny nebo v případě zapnuti komprese, odešle čísla měření. Tento způsob těží z výhody předchozího potvrzování (druhy typ), kdy jsem se zbavil potvrzování každého datového rámce a tím zrychlil komunikaci. Když by ale byla například v každém vysílání chyba, i kdyby jen jednoho datového rámce, tak je druhý způsob potvrzování nepoužitelný, protože by se jednotka slave snažila opakovaně poslat stejné data. Tento problém řeší poslední způsob potvrzování, tím že si žádá jen o datové rámce, které nebyly přijaty. Jeho nevýhodou je složitější implementace, jak na straně jednotky master, tak na straně jednotky slave.

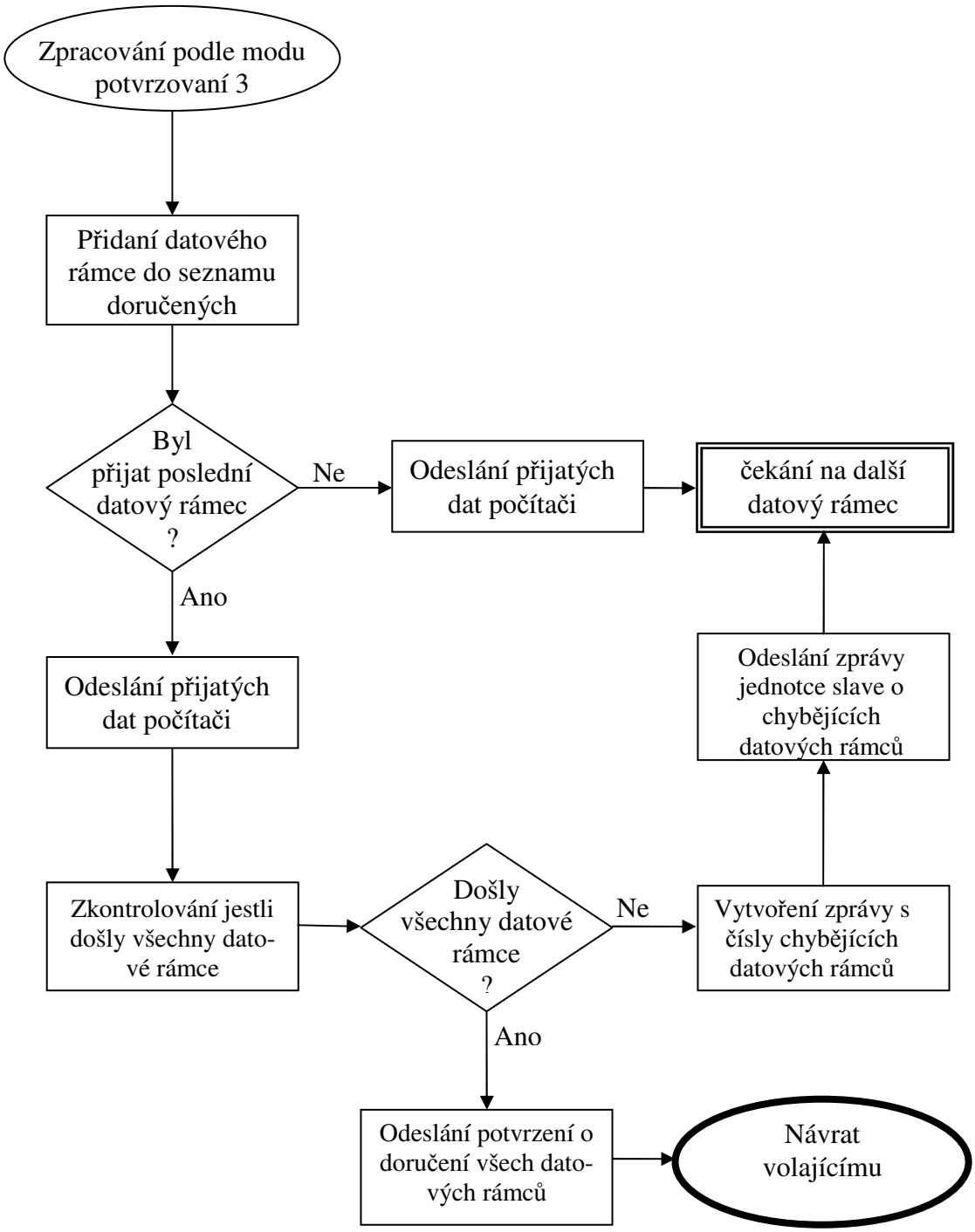
Na obr. 5.20 lze vidět zpracování přijatých datových rámců bez komprese. V prvním kroku se přidá číslo datového rámce do seznamu přijatých datových rámců. V dalším kroku se zkontroluje, jestli přijatý datový rámec je poslední, jestliže není, tak se přejde na čekání na další datový rámec. Je-li datový rámec poslední, zkontroluje se, jestli byly přijaty všechny datové rámce. Byla-li komunikace bez chyby, odešle se potvrzení o doručení všech datových rámců a odejde se z režimu potvrzování. V opačném případě se určí chybějící datové rámce a vytvoří se zpráva s čísly těchto chybějících datových rámců, které se následně pošlou, a přejde se na čekání příchodu dalšího datového rámce.

U rámců s kompresí se vypočítají chybějící datové rámce a z nich se vypočítají čísla naměřených vzorku, které nebyly doručeny. V potvrzovací zprávě se místo čísla nepřijatého datového rámce odesílají dvě čísla vzorku měření. První číslo udává první nepřijaty vzorek a druhé číslo poslední z řady nepřijatých vzorku měření, které byli v nepřijatém datovém rámci. Ostatní kroky zpracování zprávy jsou stejné jak na obr. 5.20.

Na obr. 5.19 je složení zprávy s čísly nedoručených datových rámců. Počet čísel chybějících datových rámců, které se vlezou do zprávy s čísly nedoručených datových rámců, je omezený. Jestliže počet těchto čísel je větší, tak je nutné vytvořit více zpráv. Proto tato zpráva musí nést informaci o počtu těchto zpráv a číslo zprávy, pro možné zpracování jednotkou slave.

id_zprávy	id_master	Počet datových rámců	číslo datového rámce	čísla nepřijatých datových rámců

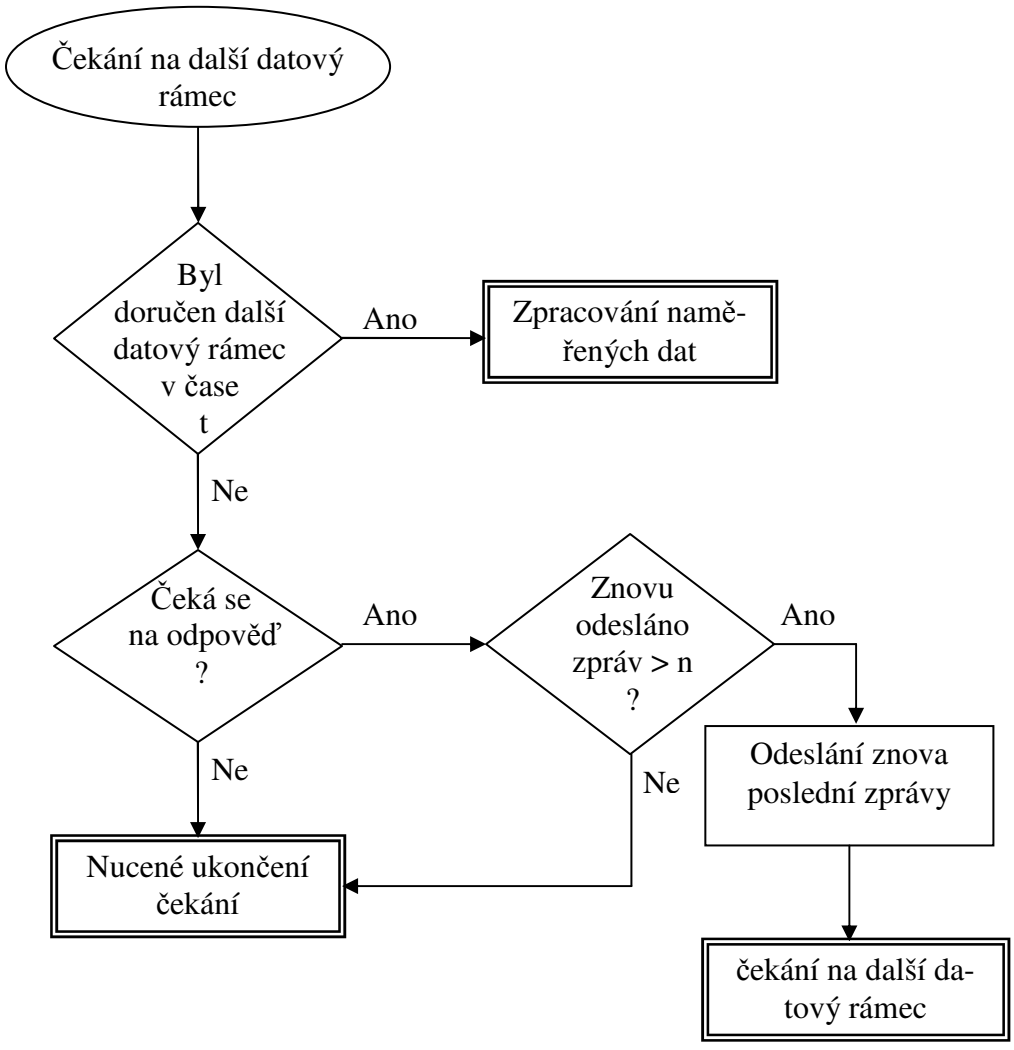
Obrázek 5.19 – Zpráva s čísly nedoručených datových rámců



Obrázek 5.20 –Diagram zpracování nových naměřených dat 3. metodou potvrzování

5.2.10. Master – čekání na další datový rámec

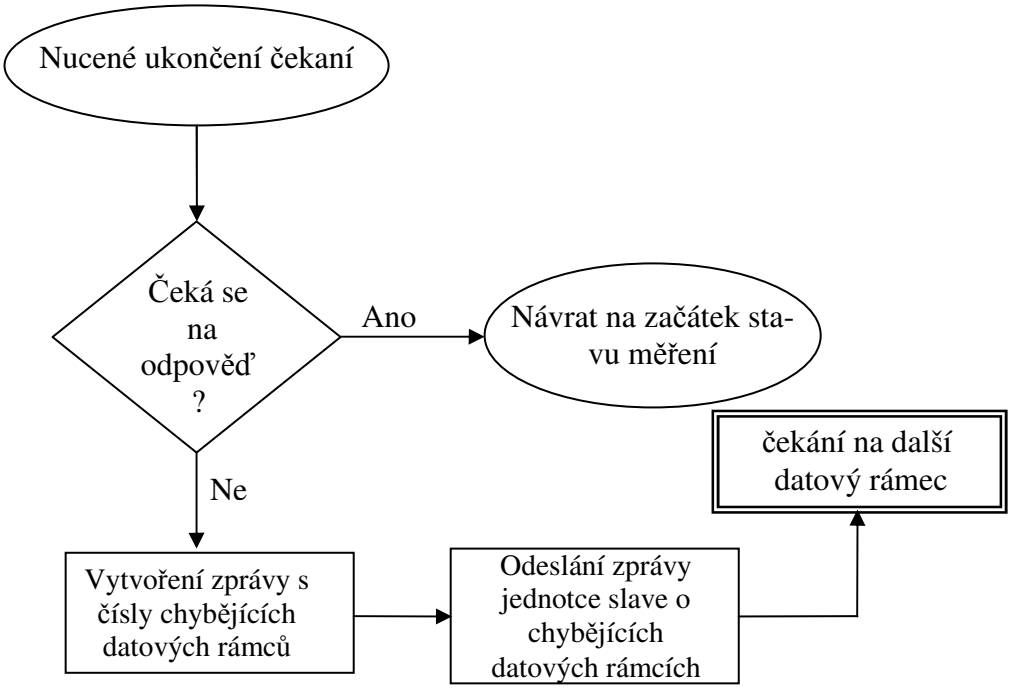
Jak jste si mohly všimnout, tak v každém potvrzovacím režimu se čeká na další datový rámec. Toto čekání musí být časově omezeno, aby nedošlo k uvíznutí a blokaci jednotky master. Než uplyne omezený časový úsek čekání, kdy nepřichází odpověď od jednotky slave, musí se jednotka master ještě několikrát pokusit poslat znovu poslední zprávu určenou jednotce slave. Na odpověď se čeká ve dvou případech. V prvním, když jsme režimu potvrzování každé zprávy, protože posíláme žádosti o poslání každého datového rámce. Ve druhém případě při posledním způsobu potvrzování, kdy žádáme o přeposlání nepřijatých datových rámců.



Obrázek 5.21 - Diagram čekání na další datový rámec

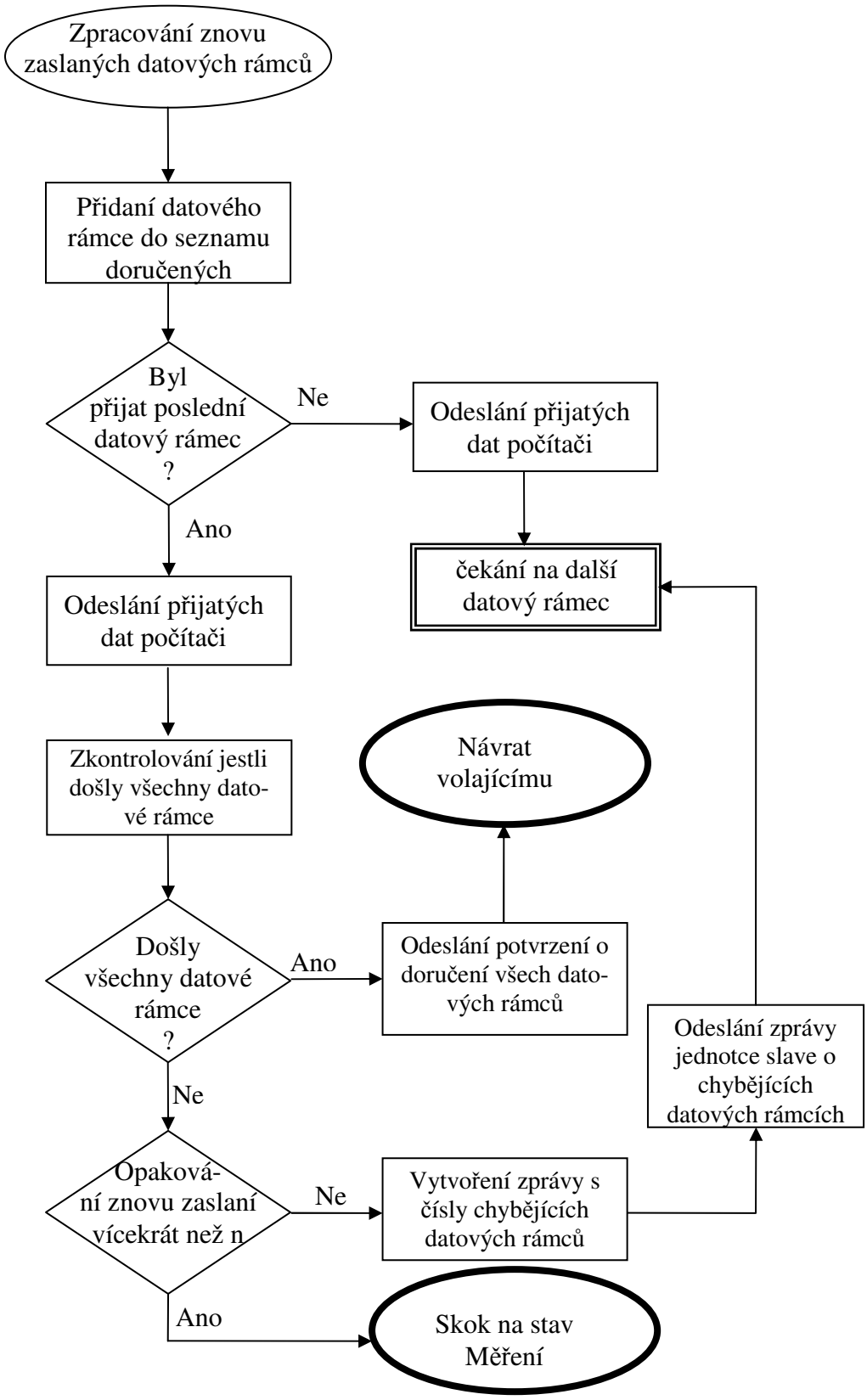
Na obr. 5.21 je popsán algoritmus čekání na další datový rámec. Když dojde datový rámec v určeném čase, pošle se na zpracování naměřených dat, jestliže nedojde, tak se další postup liší podle toho, jestli jsem čekal odpověď nebo ne. Jestliže čekám odpověď, tak přepošlu znova žádost jednotce slave za podmínky, že se toto přeposílání neopakovalo vícekrát, než je povoleno maximum.

Nejedná-li se o odpověď nebo je překročeno maximum, tak se vyvolá nucené ukončení čekání, které se liší podle zvoleného potvrzovacího režimu. Při zapnutém prvním způsobu potvrzování, se skočí na začátek stavu měření a přechází se na další jednotku slave, V případě druhého druhu potvrzování, se odešle zpráva o chybném přijetí datového rámce a následně se skočí na začátek stavu měření a přechází se na další jednotku slave. U posledního způsobu potvrzování je složitější algoritmus řešení, a proto je popsán diagramem na obr. 5.22. Rozděluje se postup podle toho, zda šlo o nepřijetí odpovědi. Nepřijetí odpovědi znamená přechod na začátek stavu měření a přechod na další jednotku slave. Nejedná-li se o nepřijetí odpovědi, tak to znamená, že nepřišel nebo nepřišly poslední datové rámce. Proto se vytvoří seznam chybějících datových rámců a odešle se jednotce slave a následuje čekání na odpověď.



Obrázek 5.22 – Diagram nucené ukončení pro 3. metodu potvrzování

5.2.11. Master - zpracování přeposlaných naměřených dat



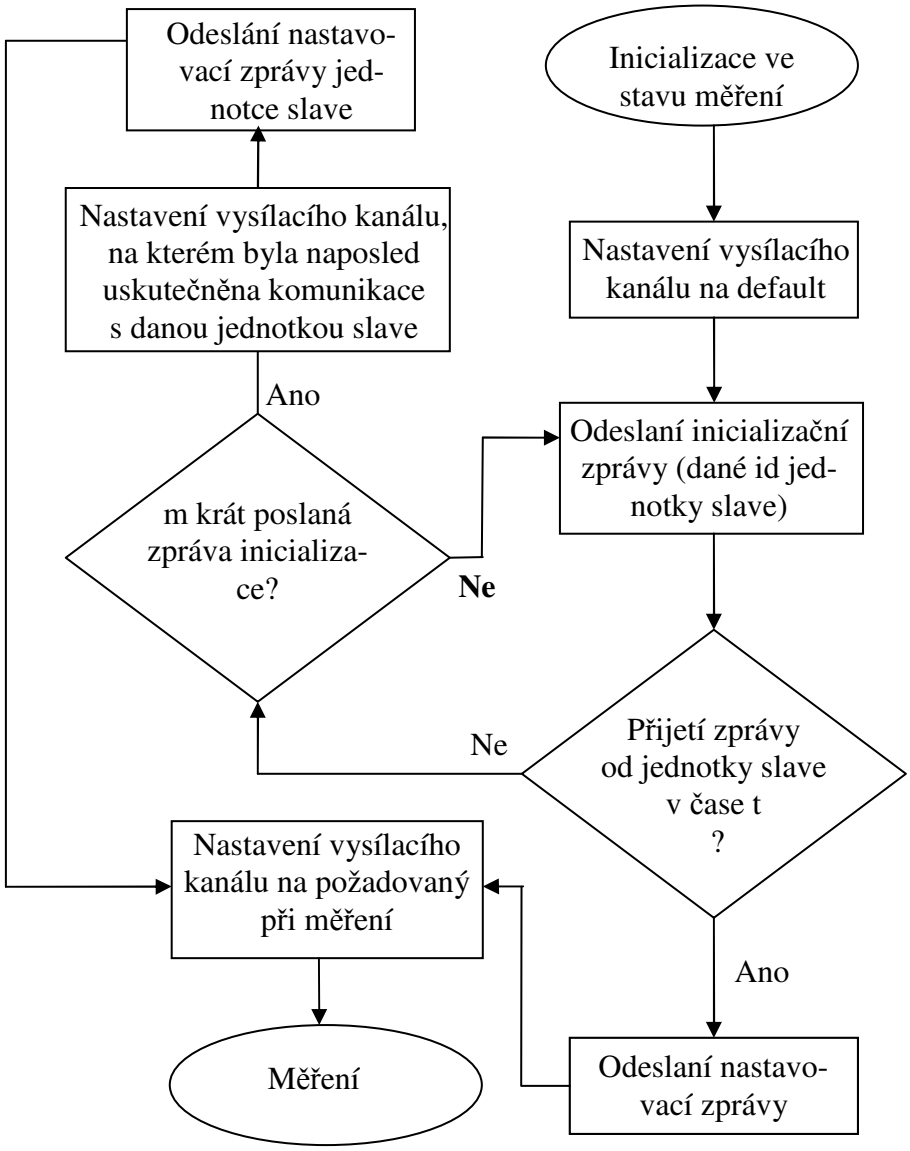
Obrázek 5.23 – Diagram zpracování přeposlaní naměřených dat

K přeposlání naměřených dat dochází jen při potvrzování přijatých datových rámců posledním způsobem. Protože prvním způsobem k přeposlání nedochází a u druhého se posílají všechny data znova (zpracovávají se jako nové).

Proto tento algoritmus vychází z třetího způsobu potvrzování. Srovnáním diagramu na obr. 5.20 a obr. 5.23 lze vidět malou změnu. Po zjištění nedoručení všech datových rámců je přidána kontrola, jestli se neopakuje dotaz o zaslání nepřijatých datových rámců vícekrát, než je maximum. Je-li větší než maximum, tak dojde k přerušení, přechodu na začátek stavu měření a výběru další jednotky slave. Jinak je algoritmus stejný.

5.2.12. Master – inicializace v rámci stavu měření

Tato inicializace nastává, jak lze vidět na obr. 5.6, po několika pokusech navázat spojení s určitou jednotkou slave. Když nastane tento stav, tak se předpokládá, že došlo k resetu jednotky slave (například výměnou baterie) nebo změně vysílacího kanálu.



Obrázek 5.24 - Diagram inicializace ve stavu měření

Na obr. 5.24je zobrazená daná inicializace. Jednotka master v prvním kroku nastaví vysílací kanál na default, na kterém přijímá jednotka slave inicializační zprávu. Poté pošle inicializační zprávu adresovanou dané jednotce slave. Jestli je do určitého času přijata odpověď od jednotky slave, tak je zaslaná nastavovací zpráva. Poté se nastaví vysílací kanál na požadovaný pro měření a vrací se zpět na začátek režimu měření. Nedojde-li v určeném čase k odpovědi, tak se navýší čítač nedoručených odpovědí a zkouší se to znova, dokud nepřijde od-

pověď nebo hodnota čítače je větší než maximum. V případě, že je hodnota čítače větší než je maximum opakování, tak se nastaví vysílací kanál, na kterém byla naposled uskutečněna komunikace s danou jednotkou slave a odešle se nastavovací zpráva. Poté se nastaví vysílací kanál na požadovaný pro měření a vrací se zpět na začátek režimu měření.

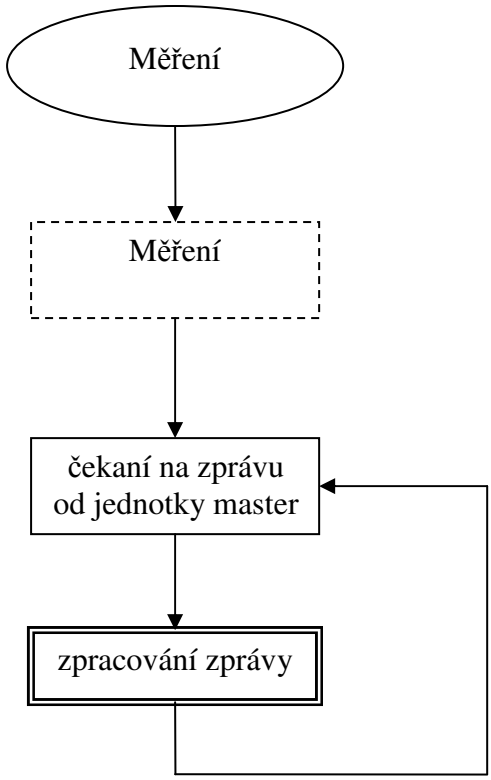
5.2.13. Master - synchronizace

Synchronizace plní dva úkoly, prvním je synchronizace času a druhým přechod všech jednotek na jiný vysílací kanál. Synchronizace času se provádí periodicky v určeném časovém úseku. Do synchronizace patří i změna vysílacího kanálu, protože jednotka master a slave musí komunikovat na stejném vysílacím kanále a touto změnou se jednoduše synchronizuje tento vysílací kanál na všech jednotkách.

Synchronizace se provádí odesláním nastavující zprávy adresované všem v bezdrátové síti. Může dojít k tomu, že tato zpráva nedojde všem jednotkám slave. Jestliže se jedna o synchronizaci času, tak jednotka slave rozezná nepřijetí nastavovací zprávy pomocí verze nastavovací zprávy, která je obsažena v žádosti o nové naměřená data. Jedna-li se o synchronizaci vysílacího kanálu, tak nebude navázán několikrát za sebou kontakt s danou jednotkou slave. Toto vyvolá inicializaci ve stavu měření, kde se pošle znova nastavovací zpráva na vysílacím kanálu, na kterém byla naposled uskutečněna komunikace s danou jednotkou slave.

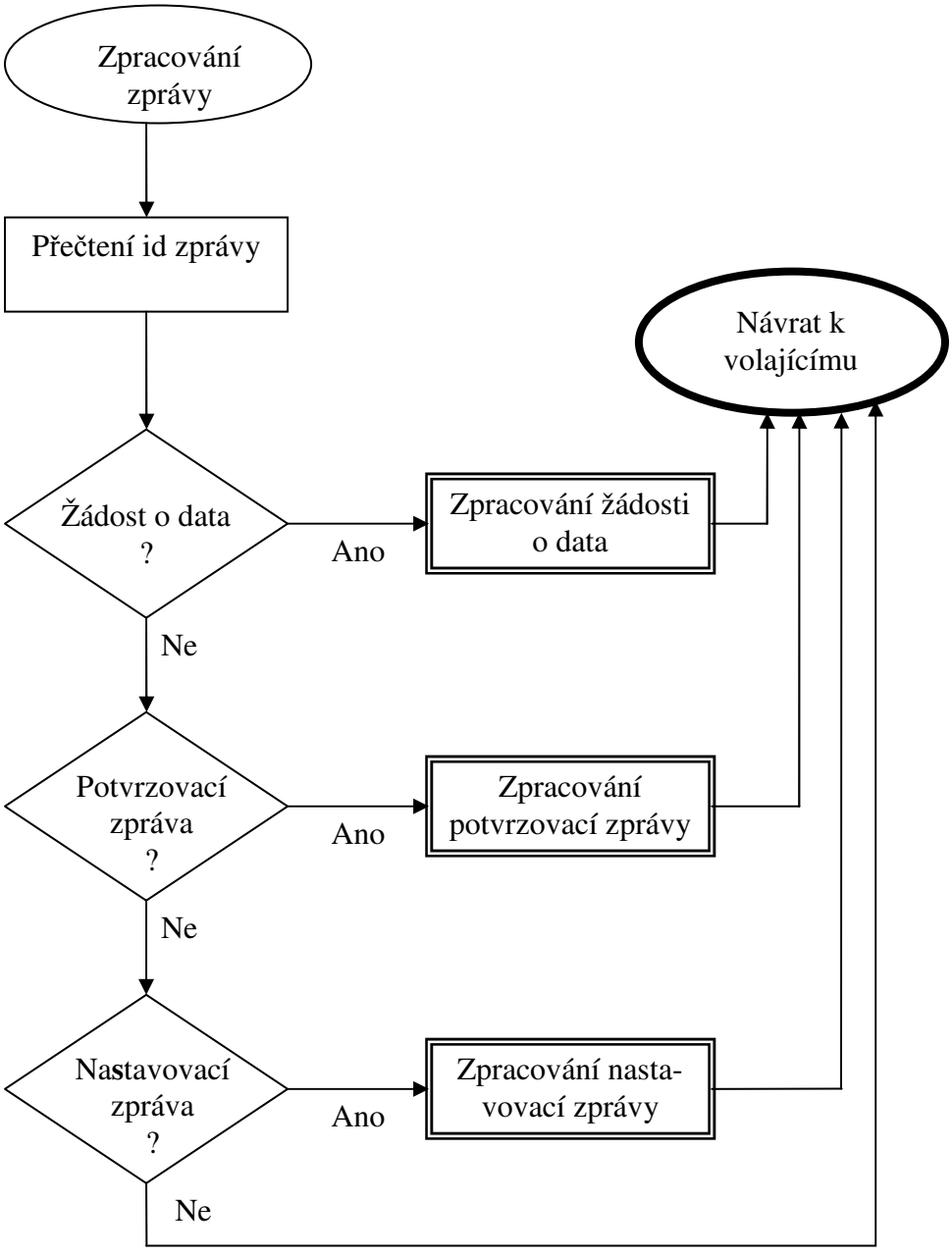
5.2.14. Slave - měřící část

Do měřícího režimu se přechází z inicializace, kde se nastavilo měřicí zařízení, vysílací kanál a typ potvrzování datových rámců. Na obr. 5.25 je zobrazen postup kroků v tomto režimu. Jelikož jednotka slave musí poslouchat jednotku master, tak dokud nepřijde zpráva od jednotky master, zůstává jednotka slave ve stavu čekání na zprávu od jednotky master. Po příchodu zprávy od jednotky master se tato zpráva zpracuje. Po zpracování se znova vrátí do stavu poslouchání. Měření akcelometru pracuje paralelně a ukládá naměřená data do zásobníku.



Obrázek 5.25 - Diagram režimu měření u jednotky slave

5.2.15. Slave – zpracování zprávy

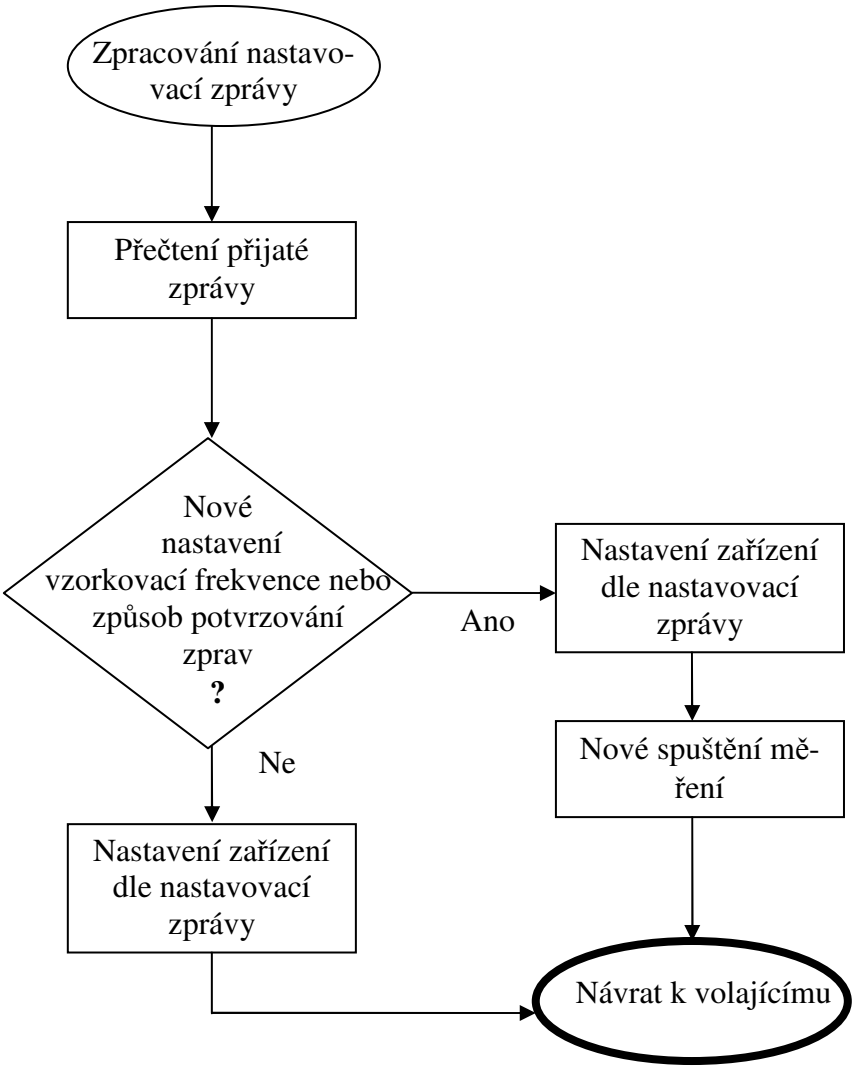


Obrázek 5.26 – Diagram slave zpracování zprávy

Po přijetí zprávy se rozezná její typ podle id zprávy a podle toho se pak dále zpracovává. V režimu měření se zpracují jen tři druhy zpráv. První je žádost o naměřená data a samozřejmě k tomu patří i potvrzovací zpráva. Poslední přijímaná zpráva je nastavovací zpráva.

5.2.16. Slave – zpracování nastavovací zprávy

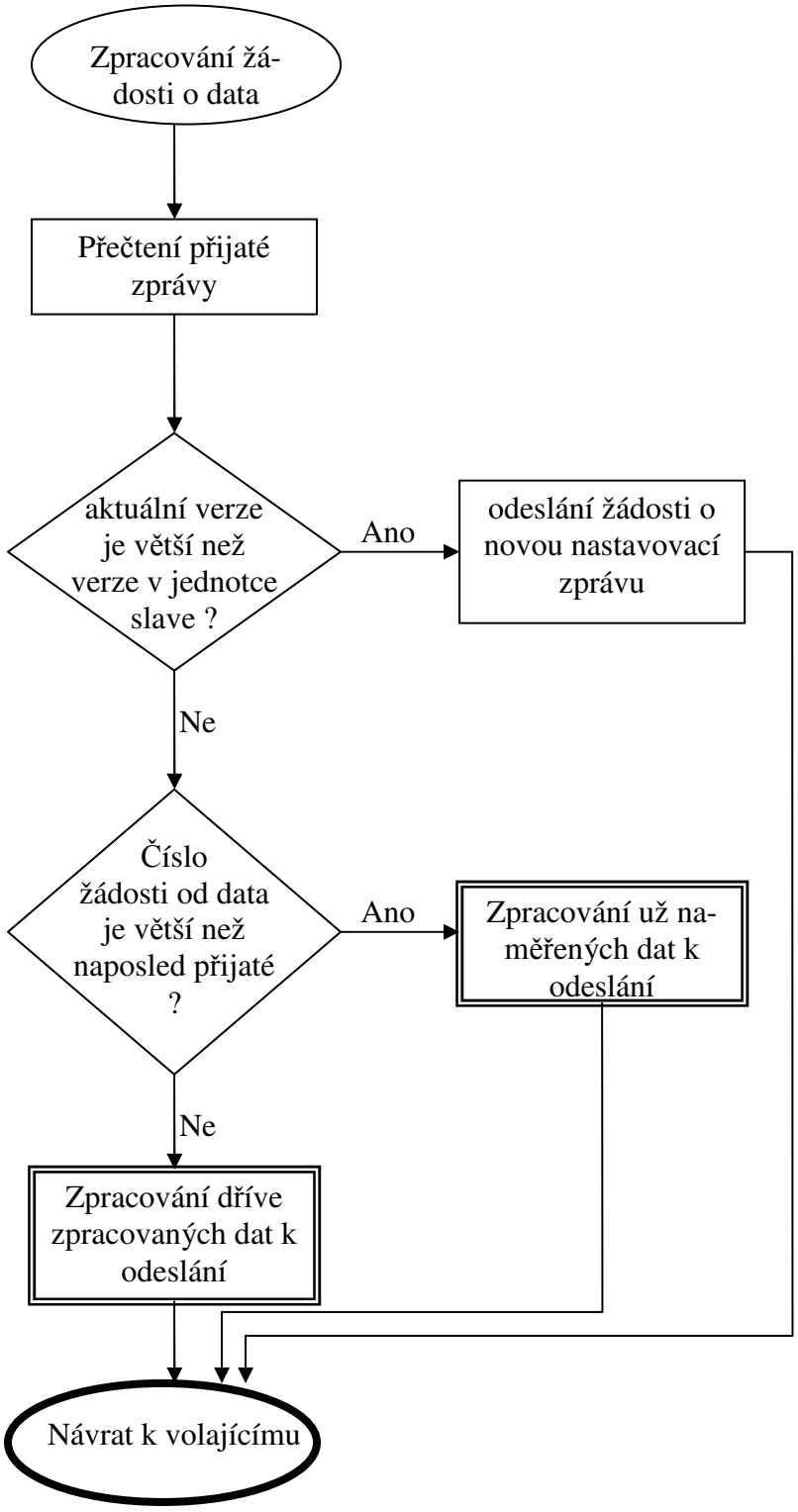
Při doručení nastavovací zprávy se přečtou hodnoty ve zprávě a podle toho se pak dále pokračuje. Nastavovací zpráva se používá k nastavení vzorkovací frekvence, způsobu potvrzování zpráv, vysílacího kanálu, komprese a synchronizace času, jejich složení je na obr. 5.2. Při nastavení nové vzorkovací frekvence nebo způsobu potvrzování zpráv se znova spouští měření, jelikož toto nastavení ovlivňuje průběh měření akcelometru. Nové spuštění měření vymaže zásobník s naměřenými daty. V ostatních případech se jen provede nastavení, bez ovlivnění průběhu měření a proto se znova nespouští měření.



Obrázek 5.27 – Diagram Slave – nastavení měřicího zařízení

Na obr. 5.27 je znázorněný postup při nastavování zařízení pomocí nastavovací zprávy.

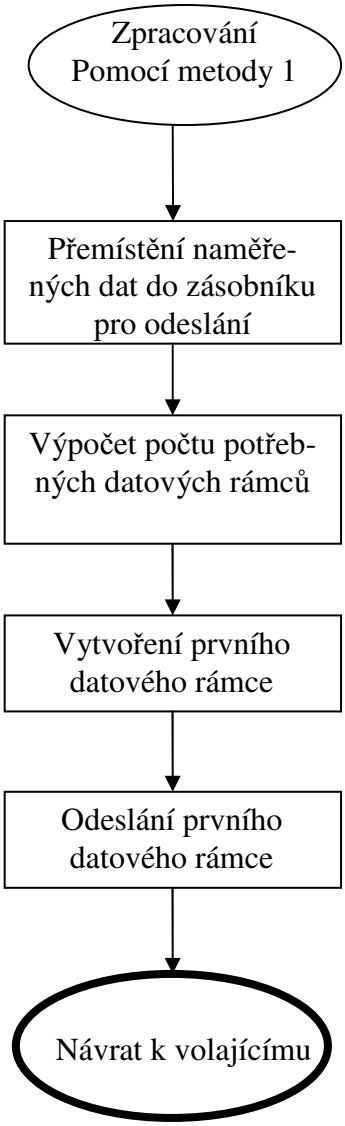
5.2.17. Slave – zpracování žádosti o data



Obrázek 5.28 – Diagram zpracování žádosti o data

Postup při zpracování žádosti o data je znázorněn na obr. 5.28, který začíná kontrolou, zda aktuální verze nastavení je stejná jako v jednotce slave. V případě že není, tak se odešle žádost o nastavovací zprávu (obr. 5.9) a vrátí se na začátek režimu měření a čeká se na zprávu od jednotky master. Má-li jednotka slave aktuální verzi nastavení, tak se porovná přijaté číslo žádosti s předcházejícím přijatým číslem žádosti. V případě, že by přijatá hodnota nebyla větší než předchozí, tak se přepokládá nepřijetí poslední odeslané zprávy a jednotka odesílá znova poslední odeslanou zprávu. Je-li přijaté číslo žádosti větší, tak jednotka přechází na zpracování naměřených dat, které dle režimu potvrzování odešle datový rámec nebo datové rámce. Po ukončení odesílaní se vrátí na začátek režimu měření a čeká na zprávu od jednotky master.

5.2.18. Zpracování naměřených dat pomocí prvního typu potvrzování zpráv



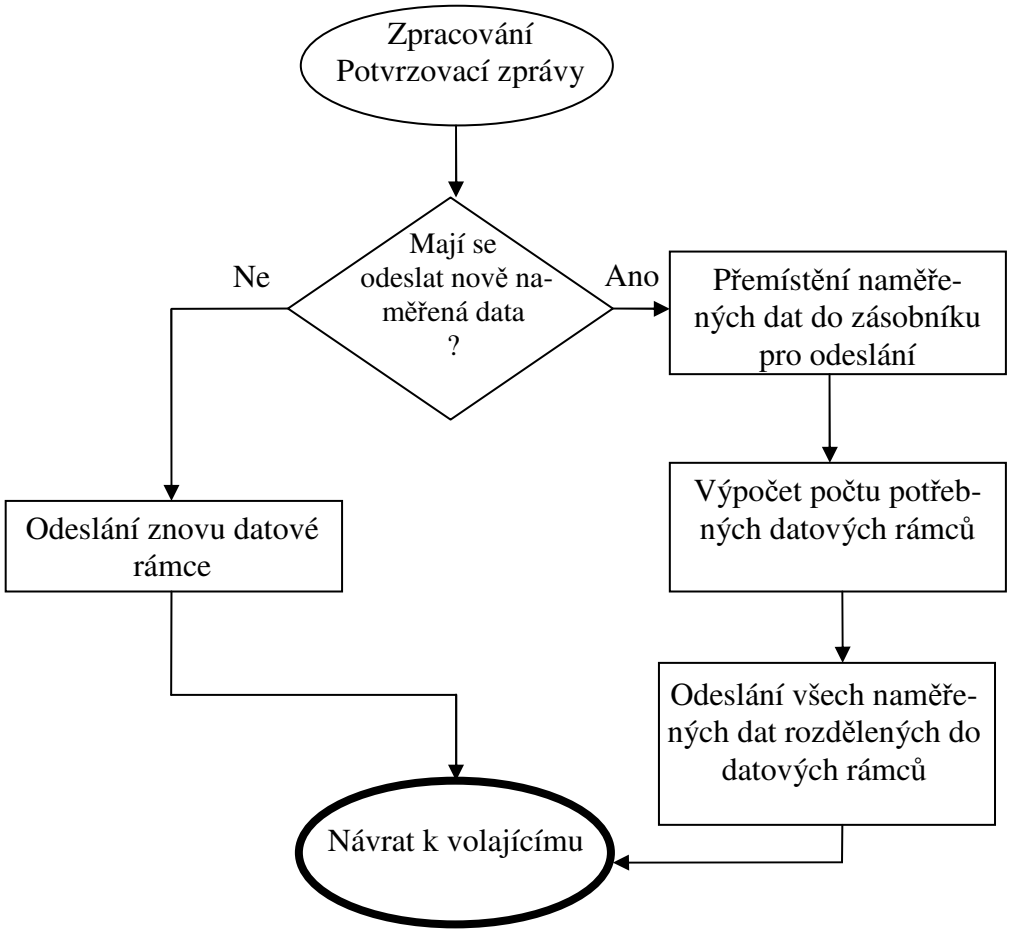
Obrázek 5.29 – Diagram Slave – zpracování naměřených dat pomocí prvního způsobu potvrzovaní

U prvního způsobu potvrzování zpráv se potvrzuje každý odeslaný datový rámec. Proto u žádosti o naměřená data se pošle jen první datový rámec a pak se čeká na potvrzení. Obr. 5.29 popisuje tento postup. Nejprve se přemístí naměřena data ze zásobníku měření do zásobníku pro odeslání a tím vyprázdníme data ze zásobníku měření pro nové měřená data. Po té se vypočítá, kolik se musí vytvořit datových rámců, pro odeslání všech naměřených dat. Pak se vytvoří první datový rámec (obr. 5.13) a odešle se jednotce master. Po odeslání se vrátí jednotka slave do začátku režimu měření a čeká na zprávu od jednotky master.

Při zapnutí komprese, se vyřízení žádostí o data nemění, jediným rozdílem je, že nelze spočítat potřebný počet datových rámců.

5.2.19. Zpracování naměřených dat pomocí druhého a třetího typu potvrzování zpráv

Jelikož třetí metoda vychází z druhé, tak tato část, tykající se odpovědi na žádost o nová data, je pro ni stejná, jen platí, že se má vždy odeslat nové data (viz. kap.).

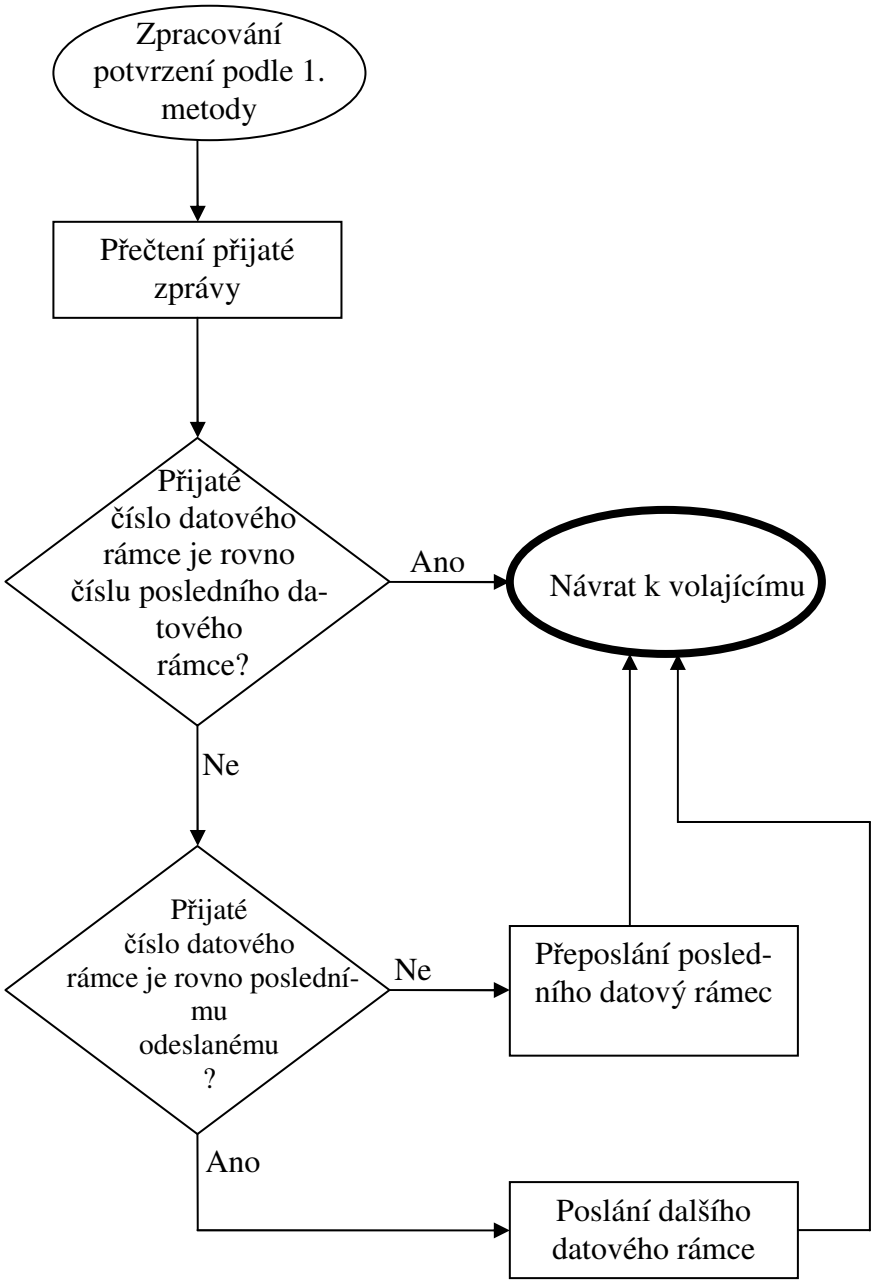


Obrázek 5.30 - Diagram zpracování naměřených dat pomocí 2. Metody potvrzování

Na začátku algoritmu (obr. 5.30) se kontroluje, jestli se mají posílat nové naměřená data. Nové data se neposílají v případě, že nedošlo potvrzení o správném doručení všech datových rámců a pošlou se znova všechny datové rámce jednotce master. V opačném případě je postup stejný jak u prvního způsobu potvrzování, jen s rozdílem, že se neposílá jen první datový rámec, ale pošlou se najednou všechny (obr. 5.29) a až po tom se přejde na začátek měření a čeká se na zprávu od jednotky master.

Při zapnutí komprese, se vyřízení žádostí o data nemění, jediným rozdílem je, že nelze spočítat potřebný počet datových rámců. Způsob odesílaní dat s kompresí, bude vysvětlen později.

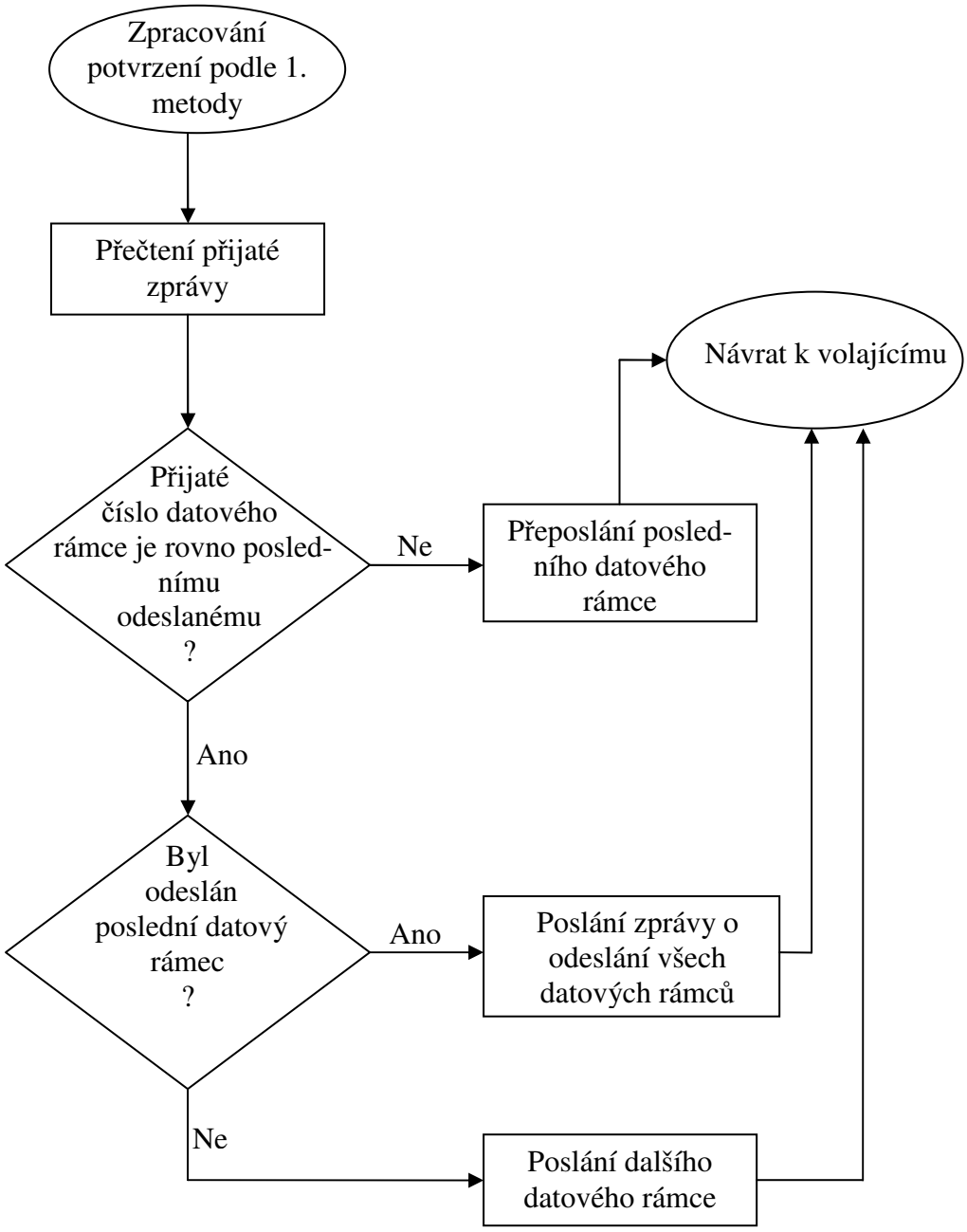
5.2.20. Slave - vyřízení potvrzení podle první metody



Obrázek 5.31- Diagram Slave – zpracování potvrzovací zprávy bez komprese podle první metody

U první metody potvrzování se potvrzuje každý odeslaný datový rámec. Zpracování tohoto potvrzení bez komprese, lze vidět na obr. 5.31. Z potvrzovací zprávy obr. 5.12 zjistíme, který datový rámec byl naposled přijat. V prvním vyhodnocení se provádí kontrola, zda je poslední přijatý datový rámec i posledním z celku datových rámců. Je-li poslední, tak se ukončí zpracování potvrzovací zprávy od jednotky master, přejde se na začátek režimu měření a čeká se na zprávu od jednotky master. Není-li přijatá zpráva potvrzení posled-

ního datového rámce z celku datových rámců, tak se provede kontrola, zda potvrzuje jednotka master přijetí posledního odeslaného datového rámce jednotkou slave. Jde-li o potvrzení posledního poslaného datového rámce, tak se odešle další datový rámec. V opačném případě se odešle znova datový rámec, který nebyl potvrzen. Následně se přejde na začátek režimu měření a čeká se na zprávu od jednotky master.

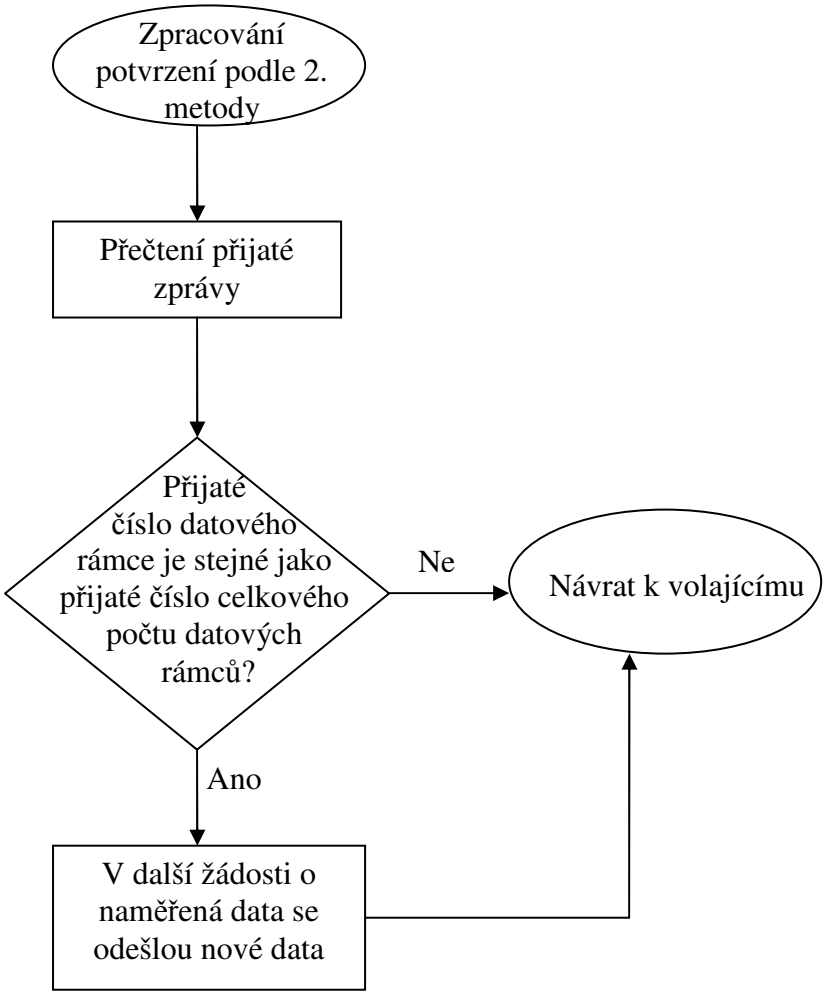


Obrázek 5.32 - Diagram Slave – zpracování potvrzovací zprávy s kompresí podle první metody

Zpracování potvrzení zpráv s kompresi je odlišné, protože předem neznáme počet datových rámců. Obr. 5.32 zobrazuje, jak postupuje zpracování potvrzení podle první metody zpráv s kompresí. Z potvrzovací zprávy se získá číslo po-

sledního přijatého datového rámce jednotkou master, které se porovná s číslem posledně poslaného datového rámce. Jestliže není stejné, dojde přeposlání posledního poslaného datového rámce, přechodu na začátek režimu měření a čekání na zprávu od jednotky master. Přijala-li jednotka master poslední odeslaný datový rámec, tak se zkontroluje, jestli tento rámec byl poslední. Není-li datový rámec poslední, tak se odešle další datový rámec, je-li poslední, odešle se zpráva o odeslání všech datových rámců. Následně se přejde na začátek režimu měření a čeká se na zprávu od jednotky master.

5.2.21. Slave - vyřízení potvrzení podle druhé metody

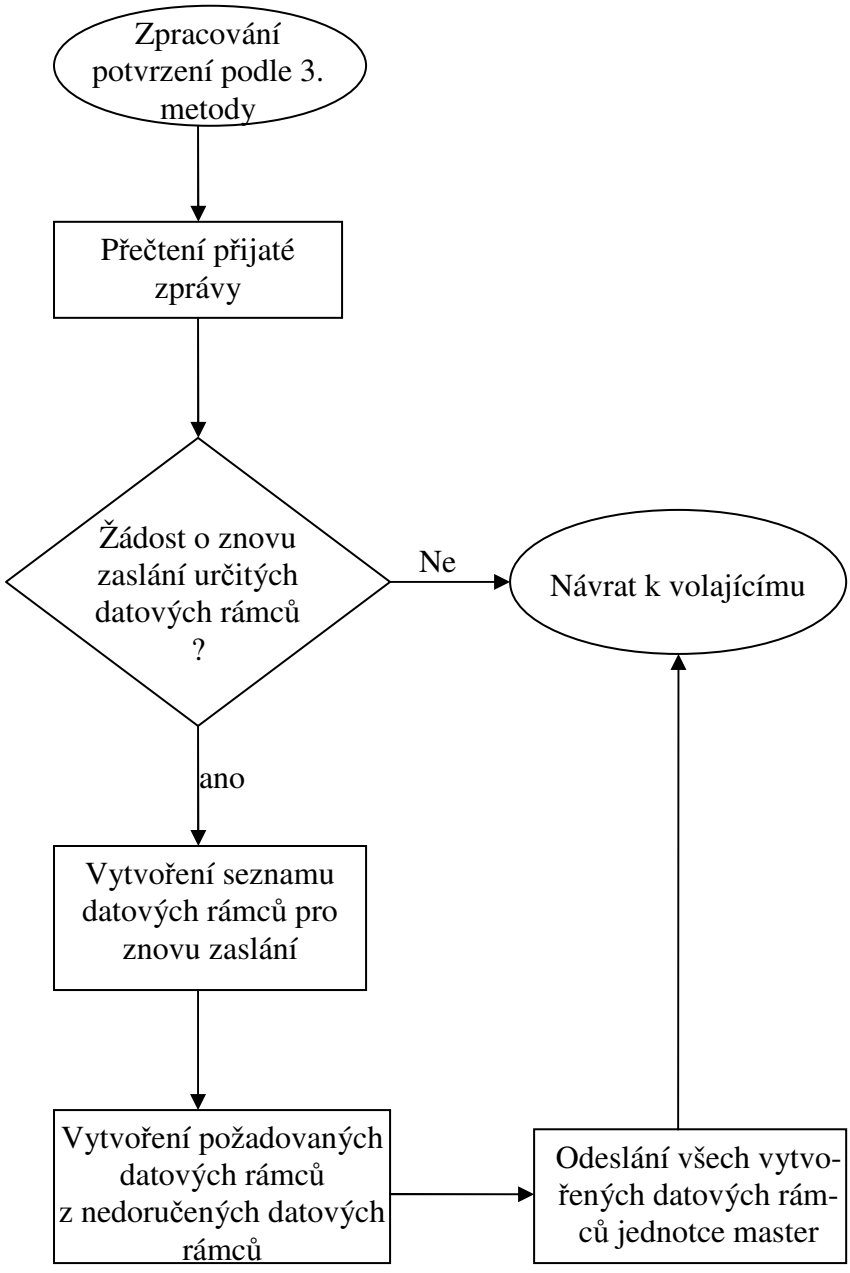


Obrázek 5.33 -Diagram Slave – zpracování potvrzovací zprávy podle druhe metody

Při druhém způsobu potvrzování jednotka slave pošle všechny datové rámce najednou a pak čeká na potvrzení od jednotky master. Jednotka master potvrzovací zprávě potvrdí přijetí všech datových rámců nebo naopak. Na obr. 5.33 je zobrazen postup zpracování potvrzovací zprávy u nekomprimovaných i

komprimovaných dat. Potvrzení přijetí všech datových rámců, je v potvrzovací zprávě obr. 5.12, řešeno tak, že číslo datového rámce a počet datových rámců se rovná. Rovnají-li se, tak se v další žádosti o naměřená data pošlou nové naměřená data.

5.2.22. Slave - vyřízení potvrzení podle třetí metody



Obrázek 5.34 - Diagram Slave – zpracování potvrzovací zprávy podle třetí metody

master a ta odešle zpět potvrzovací zprávu, ve které jsou uvedeny chybějící datové rámce (v případě dat s kompresí jsou uvedena čísla vzorku měření). Obr. 5.34 zobrazuje zpracování potvrzovací zprávy (tato zpráva je na obr.

5.19). Obsahuje-li zpráva čísla nedoručených datových rámců, tak si jednotka slave vytvoří seznam požadovaných datových rámců, které byly před tím odeslány. V případě dat s kompresí se vytvoří požadované datové rámce z nedoručených vzorku měření. Z těchto nových datových rámců se vytvoří nový celek datových rámců a odešle se jednotce master. Přejde na začátek režimu měření a čeká na zprávu od jednotky master. Došly-li všechny datové rámce jednotce master, tak se neprovádí žádné akce, přechází se na začátek stavu měření a čeká se na zprávu od jednotky master.

6. Realizace komunikačního protokolu

Jako jednotku master využívám jeden přístroj TelosB, který je připojený přes sériové rozhraní USB do počítače. Jednotky slave jsou přístroje TelosB, které mají přídavný modul s akcelometrem. Pro realizaci jsem vybral přídavný modul s analogovým akcelometrem, pro jeho jednoduší implementaci do komunikačního protokolu. S digitální akcelometrem bych musel řešit arbitraci rozhraní rádia a I2C, což by zpomalilo komunikaci mezi jednotkou slave a master. Realizovaný komunikační protokol je možné rozšířit i pro tento akcelometr.

6.1. Použité komponenty

V realizaci se musí použít různé komponenty, které poskytují například časovač, lokální hodiny, tlačítka, rádiovou komunikaci atd. V této podkapitole se zmíním o komponentech, které byly použity a nebyly zmíněny už dříve.

6.1.1. Komponent ActiveMessageC

Tento komponent je využíván pro ovládání rádiové komunikace, poskytuje řadu komunikačních rozhraní [15]. V mé práci využívám rozhraní RadioControl, RadioReceive, RadioSend, RadioPacket. Tyto rozhrání nabízí adresování zpráv pomocí id zařízení a id skupiny. Například příkaz k odeslání zprávy vypadá následovně:

call RadioSend.send[id_group](id_mode], &msg, sizeof msg);

A k přijímání události:

*event message_t *RadioReceive.receive[id_group](message_t *msg, void *payload,uint8_t len) {}*

K vytvoření zprávy používám rozhrání RadioPacket příkazem:

call RadioPacket.getPayload(&msg, NULL);

Tento příkaz vytvoří zprávu ve struktuře message_t [16]. Před použitím se musí pomocí rozhraní RadioControl radiová komunikace zapnout příkazem:

call RadioControl.start();

Další rozhraní pro práci s rádiovou komunikací nevyužívám.

6.1.2. Komponent SerialActiveMessageC

Komponent SerialActiveMessageC plní stejnou úlohu jak ActiveMessageC, s tím rozdílem, že poskytuje ovladače pro sériovou komunikaci. Většinou slouží sériové rozhraní k propojení s počítačem, aby mohl přijímat nebo odesílat data z nebo do bezdrátové sítě. Proto TinyOS 2.x zavedl formát zpráv tak, aby dovoloval multiplex i u sériového rozhraní. Díky tomu se zjednodušuje přemostění zpráv ze sériového rozhraní na rádiové rozhraní nebo naopak. Využil jsem rozhraní SerialControl, UartSend, UartReceive, UartPacket a UartAMPacket. Jelikož formát zprávy je stejný jako u rádiového vysílání, tak i příkazy a události jsou stejné [17].
Zde můžete vidět příklad:

call UartSend.send[id_group](id_mode, & msg], len)
*event message_t *UartReceive.receive[id_group](message_t *msg, void *payload, uint8_t len) {}*
call UartPacket.getPayload(&zprava2,NULL), &timeM, sizeof timeM)

6.1.3. Komponent CC2420ControlC

Tento komponent se využívá pro přístup k rozhraní CC2420Config, které poskytuje přístup k nastavení rádia. Z tohoto rozhraní využívám příkazy k přepínání vysílacích kanálu setChannel. Pro nastavení požadovaného vysílacího kanálu, se musí po zavolání příkazu setChannel, volat příkaz sync, který vyvolá změnu nastavení rádia.
Ukázka příkazu:

call CC2420Config.setChannel(DEFAULT_CHANNEL);
call CC2420Config.sync();

6.1.4. Komponent CounterToLocalTimeC

Komponent se využívá ke zjištění lokálního času v zařízení. Tento lokální čas udává časový úsek od zapnutí zařízení. Aby komponent mohl fungovat správně, je třeba propojit komponent CounterToLocalTimeC s komponentem CounterMilli32C, který poskytuje Count [18].

CounterToLocalTimeC.Counter -> CounterMilli32C;

Lokální čas je použit pro synchronizaci. Hlavní čas je lokální čas jednotky master a jednotky slave se podle něho synchronizují. Příkaz pro přečtení lokálního času:

call LocalTime.get();

6.1.5. Komponent TimerMilliC

TimerMilliC je komponent, který umožňuje různé operace s časem. Nabízí několik druhů časovačů. Základní jsou startPeriodic a startOneShot, komponent ještě nabízí rozšiřující časovače, které naleznete [19]. Při volání časovače startPeriodic se zadává čas, který bude periodicky vyvolávat událost fired v zadaném časovém úseku. U časovače startOneShot se po zadaném čase vykoná událost fired.

Při spojování s komponentem se zadává parametr, který udává jednotku času. TinyOS nabízí tři jednotky času TMilli, T32khz, and TMicro. TMilli udělá za sekundu 1024 úderů (přibližně ms), T32khz udělá za sekundu 32768 úderů, TMicro udělá za sekundu 1048576 úderů (přibližně μs). Velikost parametru je 32bitu [19].

Použité příkazy na volání časovače:

call Timer.startPeriodic(wait_interval);
call Timer.startOneShot(wait_interval);

Vzhled události:

event void TimerInc.fired(){}

6.1.6. Komponent UserButtonC

Tento komponent nabízí rozhraní pro ovládání tlačítek. Rozhraní poskytuje dva příkazy Get a Notify. Voláním Get získáme okamžitý stav tlačítka (stlačeno, nestlačeno) a pomoci Notify zapneme nebo vypneme vyvolání událostí Notify při změně stavu tlačítka.
Využití příkazu a události Notify:

call Notify.enable();
event void Notify.notify(button_state_t state) {}

6.2. Komprese

V práci se snažím zrychlit komunikaci nebo zmenšit velikost datových rámců. Jedním z řešení je komprese, díky které docílím zmenšení množství dat a tím zmenšení počtu datových rámců a následně zrychlení komunikace mezi jednotkami master a slave.

Prvním krokem při výběru komprese je rozbor dat, se kterými pracuji. V mé práci jsou to data z akcelometru, které udávají zrychlení. Při klidovém stavu pacienta nebo normálním pohybu se hodnoty mění v malém rozmezí. Přepokládám, že tyto stavy budou majoritní a stavy, kdy pacient bude mít nepřirozený pohyb, budou minoritní.

Pro práci s daty, které mění své hodnoty v určitém rozhraní, jsem zvolil typ komprese delta. Tento typ komprese uloží první hodnotu a zbylé hodnoty vyjadřuje jako rozdíl z první hodnoty [21][22].

Příklad:

Vstupní data: 0xAA 0xAC 0xAF 0x0A 0xBA 0xA1 0xA7 0xA9
Výstupní data: 0xAA 0x2 0x5 -0xF 0xF -0x9 -0x3 -0x1

6.2.1. Realizace komprese

Jedno měření akcelometru obsahuje čtyři hodnoty osy x, y, z a čas měření, každá hodnota má velikost dvou bajtů. První vzorek měření se ukládá jako výchozí hodnota, zbylé vzorky pak budou vyjádřeny jako rozdíl od výchozí hodnoty a budou mít velikost jeden bajt.

Velikost rozdílu je limitována jedním bajtem, což dovoluje maximální rozdíl 256. Tento maximální rozdíl zvětším na dvojnásobek přidáním rozlišení, zda je rozdíl kladný nebo záporný.

Jako první datový rámec se odesílá nekomprimovaná data, které obsahuje hodnoty, ze kterých se vychází při kompresi. Tento datový rámec má stejné složení jako datové rámce bez komprese obr. 5.13, kde první vzorek měření je výchozí pro kompresi a druhý je následující vzorek. Druhý vzorek se odesílá z důvodu využití celé zprávy. Další datový rámec je už s kompresí obr. 6.1. Skládá se z id zprávy, id jednotky slave, čísla datového rámce, čísla posledního odeslaného měřicího vzorku, pomocné hodnoty pro zjištění, jestli jsou rozdíly kladné nebo záporné, a nakonec data po kompresi.

id_zprávy	id_slave	číslo datového	číslo měření	a	b	c	d	data po kompresi

6.1– Naměřená data s kompresí

Do jednoho datového rámce s kompresí se vlezou 4 vzorky, každý vzorek obsahuje 4 hodnoty, takže se odesílá 16 rozdílových hodnot z výchozí hodnoty. Ke každé rozdílové hodnotě se musí přiřadit znamínko, jestli je rozdíl kladný nebo záporný. Ve zprávě to je vyřešeno pomocnými hodnotami a, b, c, d. Každá pomocná hodnota náleží k jednotlivým vzorkům. Jejich hodnota je v rozmezí 0 – 7 a každá hodnota přiřazuje znamínko plus nebo mínus k hodnotám rozdílu hodnot vzorku od výchozích hodnot vzorku. Určení, jakou hodnotu bude mít tato pomocná hodnota, popisuje tabulka 6.1.

V prvním sloupci tabulky je číslo pomocné hodnoty, další sloupce přiřazují k této hodnotě znamínka rozdílovým hodnotám. U času je vždy plus, protože vzorky jdou časově za sebou a tím pádem bude rozdíl vždy kladný.

pomocná hodnota	osa x	osa y	osa z	čas
0	+	+	+	+
1	-	+	+	+
2	+	-	+	+
3	-	-	+	+
4	+	+	-	+
5	-	+	-	+
6	+	-	-	+
7	-	+	-	+

Tabulka 6.1 – Přiřazení znamínek k pomocné hodnotě

Během komprese může nastat situace, kdy se rozdílová hodnota dostane nad maximální rozdílovou hodnotu. V tomto případě se přepíše výchozí vzorek vzorkem, který nelze zkomprimovat a odešle se datový rámec bez komprese s tímto vzorkem a vzorkem, co po něm následuje a cyklus komprese pokračuje dál s novou výchozí hodnotou.

Maximální dosažitelná komprese tohoto algoritmu je 50%.

6.3. Uspávání senzoru

Pro úsporu baterii se využívá uspávání senzorů, jestliže se jejich naměřené hodnoty pohybují určitou dobu v určitém rozmezí hodnot. Pro tuto funkci jsem využil kompresi, protože pomocí hodnoty komprese pro určitý počet vzorku, můžu určit, že se hodnoty dlouhodobě pohybují v daném rozmezí hodnot.

Jednotka master kontroluje průběžně hodnotu komprese a když rozhodné o uspání jednotky slave, tak ji pošle příkaz k uspání. Jednotka slave vypne rádiovou komunikaci pro úsporu energie, ale dál pokračuje v měření. Jednotka slave při měření kontroluje měřené hodnoty, jestli nedošlo k výkyvu nad povolené rozdílové rozmezí, což může způsobit například třepot ruky. Došlo-li k výkyvu nad povolené rozdílové rozmezí, tak senzor zapne rádiovou komunikaci, uvede se do stavu měření a začne znova odesílat naměřena data po výzvě od jednotky master. Nedojde-li po určitou dobu k výkyvu hodnot, tak se senzor probere po daném časovém intervalu sám.

Komunikace je master-slave, a proto jednotka slave nemůže po probuzení na sebe upozornit a musí čekat na zprávu od jednotky master. Proto i po uspání jednotky slave, se v každé dotazovací smyčce zkouší jednotka master dotázat dané jednotky slave, ale s tím rozdílem, že tento pokus provede jen jednou a při nedoručení odpovědi automaticky přechází na další jednotku slave.

U centrální jednotky master se úspora energie řeší jen částečně, protože se přepokládá, že má větší zásobu energie než senzory. Na rozdíl od senzoru, by měla být neustále v provozu, i kdyby byly uspány všechny senzory.

id zprávy	id master	doba uspání

Obrázek 6.2 – Zpráva k uspání jednotky slave

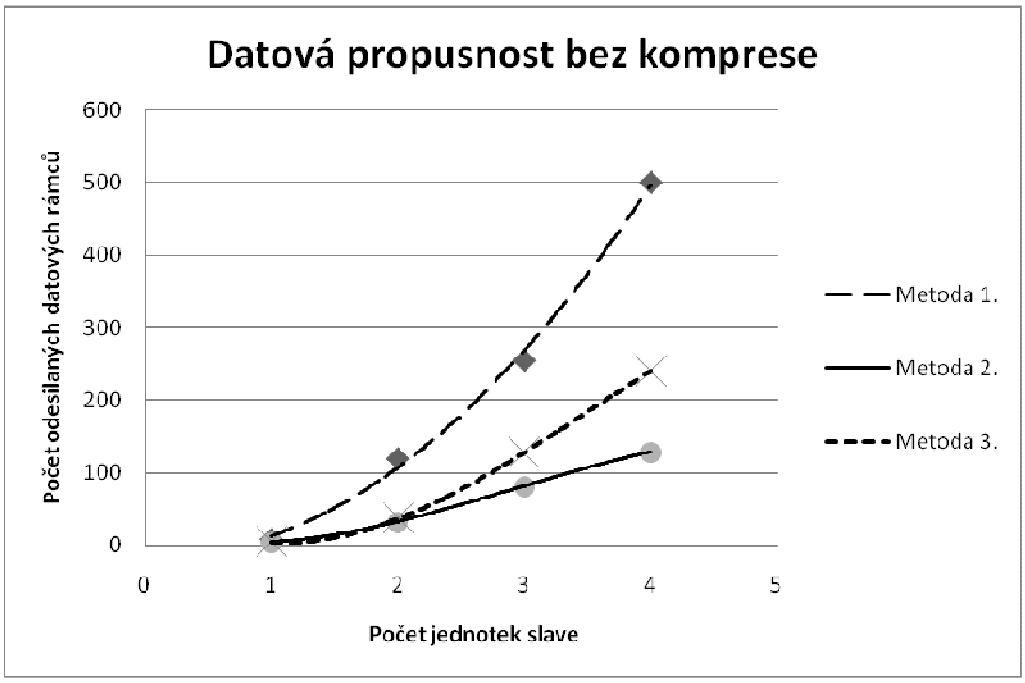
6.4. Experimenty v komunikačním protokolu

Na komunikační protokol je kladen důraz na datovou propustnost, proto jsem provedl měření zaměřené na datovou propustnost.

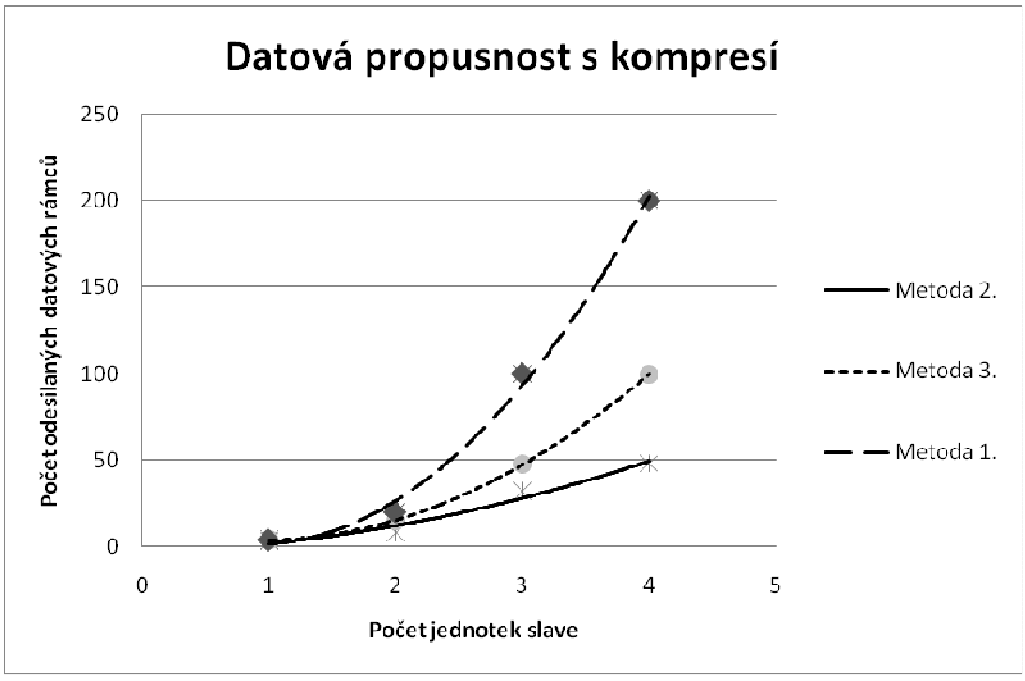
V každé metodě jsem provedl měření počtu poslaných datových rámců v jedné dotazovací smyčce v závislosti na počtu připojených jednotek slave.

Na obr. 6.3 je uvedená závislost poslaných datových rámců v jedné dotazovací smyčce při vypnutí komprese a bez chyb komunikace. V Grafu lze vidět, že při použití první metody exponenciálně narůstá počet poslaných datových rámců. Porovnáním druhé metody s první, lze vidět, že počet datových rámců stoupá pomaleji a lineárně. Druhá a třetí metoda by měla stejný počet datových rámců, kdyby třetí metoda nezatěžovala jednotku master kontrolou přijatých dat. Doba provádění algoritmu kontroly se s nárůstem počtu datových rámců zvětšuje. Dochází tak k nárůstu doby přechodu na další jednotku slave. V důsledku toho se zvýší doba trvání jedné smyčky a tím se i zvýší počet naměřených dat během smyčky. Na obr. 6.4 je uvedená závislost poslaných datových rámců v jedné dotazovací smyčce při zapnutí komprese a bez chyb komunikace. Při zapnutí komprese se zmenší počet datových rámců nutných pro odeslání všech vzorku měření, které jsou v jednotce slave. Proto i na grafu (obr. 6.3) lze pozorovat pokles datových rámců při použití komprese.

Vyplývá, že při zanedbání chyb komunikace je nejlepší druhá metoda. Budu-li uvažovat o chybách komunikace, tak se výsledky budou měnit v závislosti na chybách. Nejvíc robustní proti chybám je metoda první, která potvrzuje každé přijetí datového rámce. Metoda druhá je velmi závislá na chybách. Je-li četnost chyb tak velká, že zasahuje do každé komunikace mezi jednotkou slave a master, tak je tato metoda nepoužitelná, protože se nikdy nepovede poslat všechny datové rámce najednou. Třetí metoda je kompromis mezi robustnosti proti chybám a datové propustnosti. Experimenty byly prováděny při stejné vzorkovací frekvenci 15ms a na vysílacím kanálu 26. Zvýšením vzorkovací frekvence se zvětší počet datových rámců, protože během jedné smyčky se provede více měření a tím vznikne více vzorků pro odeslání. Změna kanálů ovlivní datovou propustnost podle zatížení toho kanálu jinými bezdrátovými sítěmi.

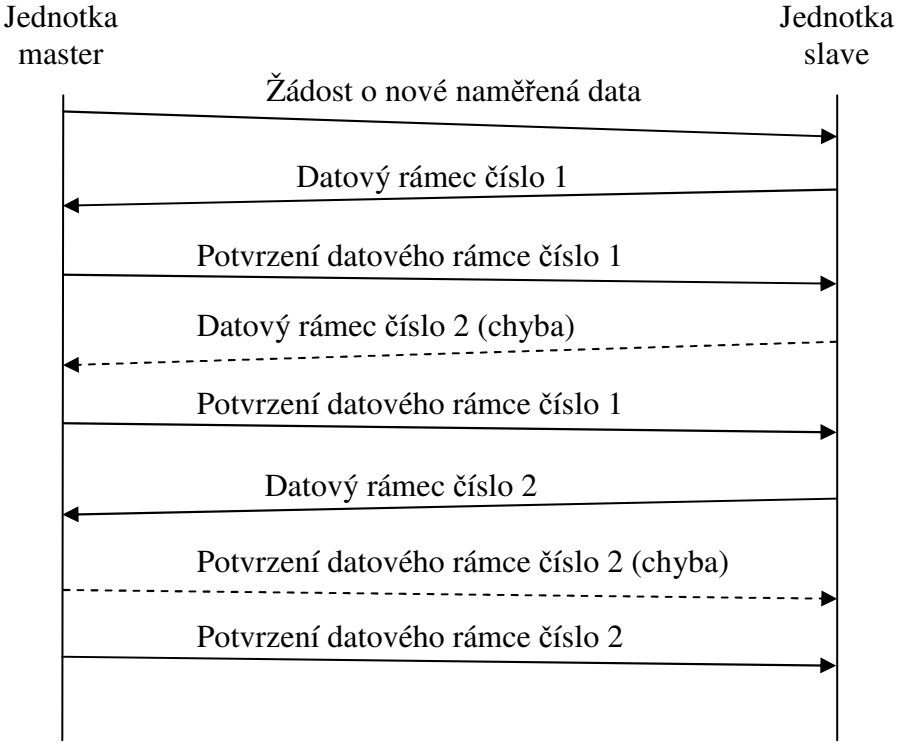


Obrázek 6.3 – Graf datové propustnosti při nepoužití komprese



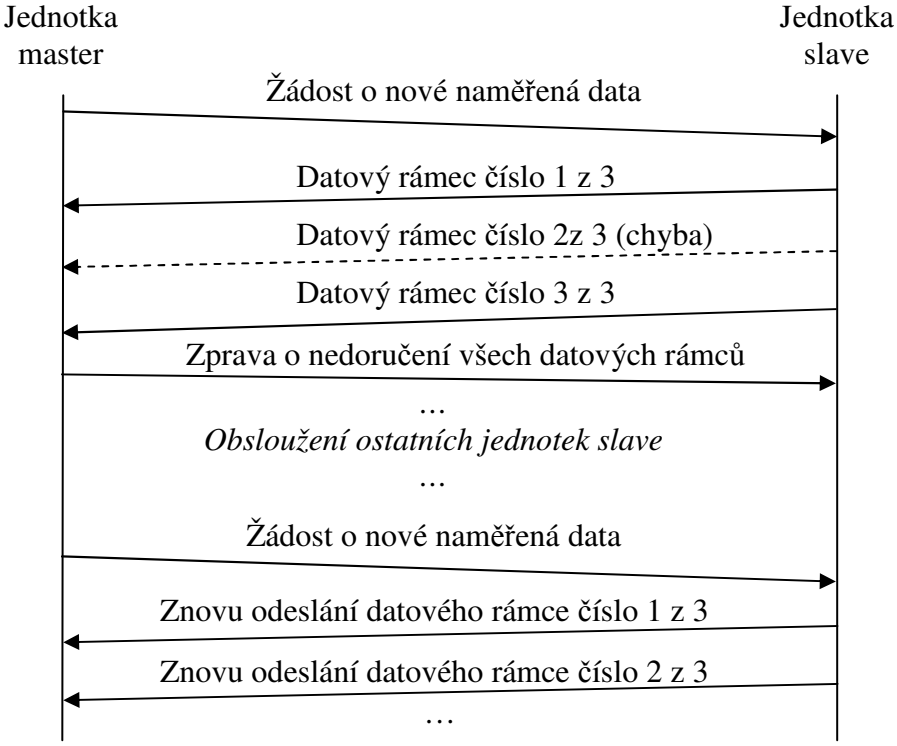
Obrázek 6.4 - Graf datové propustnosti při použití komprese

Další experiment zkoumá chování při simulaci chyby. Tento experiment jsem zobrazil pomocí diagramu.



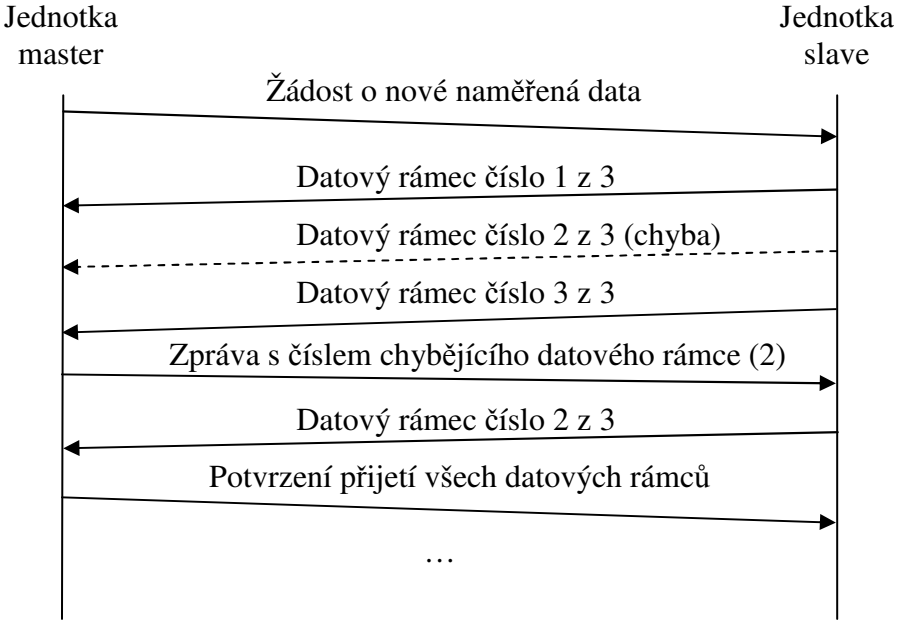
Obrázek 6.5 – Diagram simulace chyby v metodě 1.

Obr. 6.5 popisuje řešení problému nepřijetí datového rámce jednotkou master při prvním způsobu potvrzování. Z diagramu lze vidět, že při nepřijetí datového rámce jednotkou master, odešle jednotka master znova potvrzení o přijetí datového rámce, který byl naposled přijat. Díky tomu odešle jednotka slave znova nepřijatý datový rámec. Na diagramu lze pozorovat, že nedoručení dalšího datového rámce může způsobit, buď nepřijetí jednotkou slave zprávy potvrzující přijetí posledního odeslaného datového rámce nebo nepřijetí dalšího datového rámce jednotkou master.



Obrázek 6.6– Diagram simulace chyby v metodě 2.

Obr 6.6 ukazuje zpracování chyby komunikace v druhé metodě potvrzování. Při výskytu chyby jednotka master čeká na ukončení vysílání jednotky slave a až poté odešle zprávu o chybné komunikaci. Následně přejde na komunikaci s další jednotkou slave. Po obsloužení ostatních jednotek slave se vrací k jednotce, u které nastala chyba komunikace. Jednotka master této jednotce slave pošle žádost o nové naměřená data. Jednotka slave začne vysílat znova data, které v minulé komunikaci nebyly v pořádku doručeny.



Obrázek 6.7 – Diagram simulace chyby v metodě 3.

Chybu při odesílání datových rámců ve třetí metodě popisuje diagram na obr. 6.7. Při výskytu chyby čeká jednotka master na ukončení odesílaní datových rámců jednotkou slave, stejně jako u druhé metody. Po dokončení odesílání provede jednotka master výpočet chybějících datových rámců. V tomto experimentu nebyl přijat při vysílání druhý datový rámec, proto jednotka master odešle zprávu, že chybí datový rámec s číslem 2. Jednotka slave odpoví na zprávu odesláním datového rámce číslo 2.

7. Vizualizace a ukládání dat v PC

7.1. Propojení Pc a TelosB

Zařízení TelosB nabízí komunikaci s PC pomocí sériového rozhraní. Pro tuto komunikaci, využiji hotový komponent pro TelosB a hotové třídy v jazyce Java pro PC, které nabízí TinyOS. Komponent pro sériovou komunikaci jsem popsal v předchozí kapitole (kap. 6.1.2.).

Pro propojení aplikace a sériového rozhraní se využívá třída MoteIF, která poskytuje aplikační úroveň Java rozhraní pro příjem a odesílání zpráv přes sériový port z nebo do TelosB. V instanci třídy MoteIF se musí zaregistrovat MessageListener, který zpracovává příchozí zprávu. Pro odesílání zpráv se využívá metoda send.

Registrace registerListener požaduje parametr, který udává typ zprávy. Proto bylo nutné vytvořit třídu, která udává typ zprávy a obsahuje metody a atributy, které jsou potřebné pro zpracování přijaté zprávy. Jsou to například metody, které vrací: datový řetězec přijaté zprávy, přijaté naměřené data, verzi, interval…atd.

Jak už jsem psal v kapitole 6.1.2., tak sériová komunikace je navrhnuta tak, aby bylo možné přemostění zprávy ze sériového portu na rádiové vysílání. Z tohoto důvodu se metoda send volá s použitím dvou parametrů. Jeden parametr je adresa a druhý posílaná zpráva. Parametr adresa je použít, když chceme pomocí zařízení TelosB, který je připojen sériovým rozhraním k PC, přeposlat zprávu jinému zařízení v bezdrátové sítí. Já v aplikaci používám odesílanou zprávu pro nastavení nastavovací zprávy v jednotce master, proto přemostění v jednotce master nepoužívám a parametr adresa není pro mě důležitá. Do parametru zprávy se udává typ zprávy, pro který jsem musel udělat novou třídu, která dovoluje upravovat parametry v této zprávě.

7.2. Požadavky na aplikaci

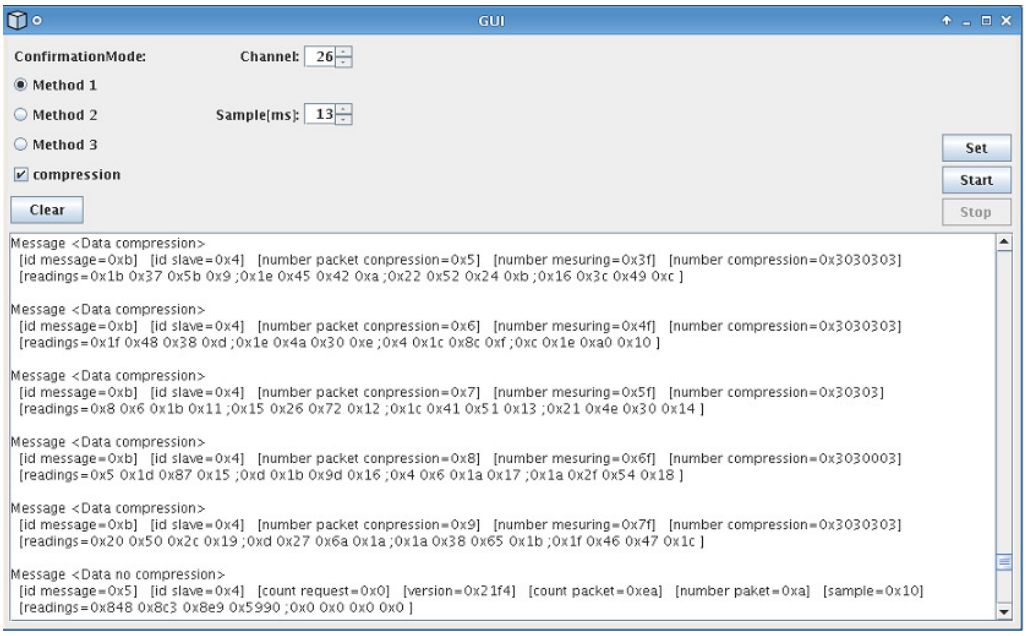
Aplikace v počítací slouží k vizualizaci a ukládání naměřených dat do souboru. Vizualizace nabízí změnu nastavení měření, způsobu komunikace a výpis příchozích dat. Ukládání do souboru slouží k archivaci naměřených dat.

7.3. Realizace Aplikace

Pro vizualizaci bylo vytvořeno okno, které obsahuje možnost nastavit metodu potvrzování, zapnutí komprese, vysílací kanál, vzorkovací frekvenci. Dále obsahuje tlačítko pro odeslání nového nastavení, tlačítko pro zapnutí výpisu, tlačítko pro vypnutí výpisu, pole obsahující výpis příchozích dat a tlačítko pro smazání výpisu. Na obr. 7.11 je zobrazen vzhled okna.

Data se ukládají do souboru. Každý vzorek naměřených dat je oddělený od dalšího vzorku středníkem. Na jednom řádku jsou uloženy data, které přišli

v jednom celku datových rámců a na následujícím řádku je uložen aktuální lokální čas v jednotce master. Ukázka zápisu do souboru je na obr. 7.2.



Obrázek 7.1 - Okno aplikace vizualizace

```
0x831 0x880 0x851 0x8d9a ;0x8bc 0x8ef 0x941 0x8d9f ;0x864 0x901 0x8a2 0x8dae ;0x846 0x8d3 0x8f4
 id slave=0x4 time=0x39956
0x839 0x8ad 0x80f 0x92e4 ;0x830 0x870 0x90a 0x92e7 ;0x880 0x86b 0x936 0x92ed ;0x84d 0x8bf 0x8c5
 id slave=0xb time=0x39c01
0x84a 0x8cf 0x8ed 0x95e8 ;0x8bd 0x8c8 0x8c7 0x95ec ;0x4 0x0 0x6 0x6 0xb2 0xb1 0x1c 0x1 ;0x22 0x
 id slave=0x4 time=0x39e69
0x7d6 0x82c 0x81c 0x9980 ;0x82e 0x879 0x8f8 0x998f ;0x0 0x0 0x0 0x0 0xc3 0x67 0x77 0x1 ;0x74 0x
 id slave=0xb time=0x3a0cb
0x7fb 0x899 0x843 0x9c25 ;0x8a4 0x8c1 0x8b0 0x9c2d ;0x0 0x0 0x0 0x0 0x64 0x71 0x60 0x1 ;0x4b 0x
 id slave=0x4 time=0x3a2da
0x8c3 0x846 0x872 0x9ee4 ;0x8c3 0x932 0x906 0x9ee6 ;0x876 0x94e 0x943 0x9ef0 ;0x857 0x90c 0x8b1
 id slave=0xb time=0x3a49a
0x7e4 0x85a 0x86f 0xa0e9 ;0x85c 0x8e3 0x90e 0xa0f5 ;0x0 0x0 0x0 0x0 0x69 0x67 0x72 0x1 ;0x79 0x
 id slave=0x4 time=0x3a62f
0x7df 0x83c 0x83b 0xa2ff ;0x832 0x872 0x8f1 0xa30e ;0x0 0x0 0x0 0x0 0x93 0xd4 0x6e 0x1 ;0x6b 0x
 id slave=0x4 time=0x3adea
0x7da 0x852 0x84d 0xa656 ;0x834 0x884 0x846 0xa662 ;0x4 0x0 0x0 0x0 0x53 0x2e 0xc 0x1 ;0x61 0x5
 id slave=0xb time=0x3b129
0x93e 0x87f 0x893 0xab81 ;0x93e 0x8af 0x8dc 0xab84 ;0x1 0x1 0x1 0x1 0x9c 0x21 0x7 0x1 ;0xf1 0x4
 id slave=0x4 time=0x3b374
0x7d6 0x849 0x82c 0xae11 ;0x82d 0x87f 0x90d 0xae1f ;0x0 0x0 0x0 0x0 0x6d 0x65 0x93 0x1 ;0x6d 0x
 id slave=0xb time=0x3b5c3
0x7f7 0x894 0x83c 0xb14c ;0x8ad 0x8df 0x928 0xb151 ;0x862 0x905 0x8af 0xb15d ;0x846 0x8ce 0x8f0
 id slave=0x4 time=0x3b7d1
0x7c2 0x825 0x817 0xb396 ;0x838 0x89a 0x875 0xb3a1 ;0x0 0x0 0x0 0x0 0x8c 0xa4 0xc7 0x1 ;0x87 0x
```

Obrázek 7.2 – ukázka zápisu do souboru

8. Závěr

V této práci jsem se musel seznámit s problematikou bezdrátové senzorové sítě pro monitorování stavu pacientu s Parkinsonovou chorobou, předchozí prací [1], na kterou navazuji, a s operačním systémem TinyOS. Dále jsem se musel seznámit s platformou TelosB a přídavnými moduly s akcelometrem.

Prvním krokem práce na bezdrátové senzorové sítí bylo vytvoření ovladačů přídavných modulů s akcelometrem pro operační systém TinyOS 2.x, protože v předchozí práci byly vytvořeny pro operační systém TinyOS 1.x a tyto ovladače se nedají použít v TinyOS 2.x.

V dalším kroku byl vytvořen návrh komunikačního protokolu pro bezdrátovou senzorovou síť typu Master – Slave, která zabezpečuje sběr dat ze senzoru do centrální jednotky master. Při vytváření komunikačního protokolu se vycházelo z požadavku na velkou propustnost dat, tak aby komunikace byla rychlá natolik, aby byl možný sběr dat, která jsou v akcelometru odebírány v určitém kmitočtu. Dalším požadavek vyžadoval metody pro úspory energie, protože energetické zdroje u bezdrátových senzorů jsou omezeny.

Po návrhu komunikačního protokolu byla provedena jeho implementace s implementací ovladačů přídavných modulu. Navržený komunikační protokol obsahuje tři metody potvrzování, možnost komprese naměřených dat a uspávaní senzorů.

Při testování datové propustnosti pro každý režim potvrzování, dopadla nejhůře první metoda, která potvrzuje každý přijatý datový rámec. Nejlépe dopadla třetí metoda, která měla největší datovou propustnost i při výskytu náhodných chyb. Tato metoda potvrzuje datové rámce, až když jsou odeslány jednotkou slave všechny. Jednotka slave odesílá při chybné komunikaci jen datové rámce nepřijaté jednotkou master.

První metoda potvrzování má jednoduché zabezpečení kontroly příchozích datových rámců, tím že potvrzuje každý přijatý datový rámec a nedovolí jednotce slave vynechat při odesílání žádný datový rámec. Nevýhodou této metody je navýšení počtu nutných zpráv i při bezchybné komunikaci.

Druhá metoda potvrzování potvrzuje přijetí kompletního datového celku. Díky tomu se zvýší datová propustnost v případě bezchybné komunikace, ale v případě kdy dochází k častým chybám, se propustnost zmenšuje. Důvodem je, že při jakékoliv chybě komunikace je odeslán požadavek o znovu zaslání všech datových rámců.

Třetí metoda potvrzuje přijetí datových rámců až po odeslání všech datových rámců jednotkou slave. Datová propustnost při bezchybné komunikaci je stejná jako u druhé metody, jen s rozdílem, že přechod na další jednotku slave je opožděn o provedení kontrolního algoritmu v jednotce master. Při výskytu chyb v komunikaci je datová propustnost lepší než u druhé metody, jelikož se neopakuje vysílání všech datových rámců, ale jen těch co nebyly doručeny.

Ze srovnání těchto metod vyplývá, že pro komunikaci, kde dochází k velmi častým chybám, je vhodná metoda první. Problém je, že jednotky slave musí mít pro tuto komunikaci velký zásobník pro naměřená data, protože je velmi malá datová propustnost. Při bezchybné komunikaci je výhodnější druhá metoda, protože zaručuje největší datovou propustnost. Dochází-li k občasným chy-

bám, tak je vhodnější třetí metoda, která má dobrou propustnost bezchybné i chybové komunikaci.

Pro kompresi dat byla zvolena metoda komprese delta, díky které bylo docíleno maximální hodnoty kompresního poměru 50%. Při vykonávání metod, kdy se zpracovávají data s kompresí, byla přidána funkce uspávání senzorů. K uspání senzoru dochází, když se poměr komprese u zvoleného počtu vzorku dostane nad zvolenou hodnotu. Uspaný senzor vypne rádio a tím zmenší spotřebu energie. Senzor provádí měření dál a při velké odchylce se znova probudí nebo se probudí po uplynutí časového úseku, zvoleného jednotkou master při uspávání jednotky slave.

Byla vytvořena aplikace pro vizualizaci, ovládaní zařízení, výpis vzorků měření a ukládání dat do souboru. Tato aplikace byla napsána v jazyce Java.

Posledním krokem bylo vytvoření live cd s Linuxem, který obsahuje prostředí TinyOS, vytvořené komponenty, aplikace v této práci a návod použití.

Další možností pokračování na této práci je rozšíření zpracování naměřených dat např. přidáním filtrace, grafické zobrazení pohybu…atd.

9. Literatura

[1] AUERSVALD M. *Wireless sensor network for monitoring patiens with Parkinson's disease.* Diplomová práce. Praha: České vysoké učení technické, katedra řídicí techniky,2008

[2] POKORNÝ P. *Parkinsonova choroba*[online]. Poslední revize 2004-06-20 [cit.2009-04 -20]. <http://www.ordinace.cz/clanek/parkinsonova-choroba/,2004>

[3] K.KARL,WILLIG A. *Protocols and Architectures for Wireless Sensor Networks.* West Sussex:Jonh Wiley & Sons Ltd,The Atrium, Southern Gate,Chichester,2005 ISBN -13 978-0470-09510-2

[4] CROSSBOW TECHNOLOGIES. *TELOSB.[online].* Poslední revise 2006-01-03[cit.2009-04 -20]. <http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf>

[5] Texas Instruments/Chipcon, *CC2420 Datasheet.* rev 2008-04-11,verze neudána <http://focus.ti.com/docs/prod/folders/print/cc2420.html>

[6] Texas Instruments, *MSP430F1611 Datasheet.* Rev F 2009-5-18 <http://inst.eecs.berkeley.edu/~ee100/fa08/lab/project/MMA7260Q-Rev1.pdf>

[7] FTPI chip, *driver[*online*]*,2009 <http://www.ftdichip.com/FTDrivers.htm >

[8] STMicroelectronics, *LIS3LV02DQ Datasheet.* Rev 1 2005, CD00047926 <http://www.alldatasheet.com/datasheet-pdf/pdf/117096/STMICROELECTRONICS/LIS3LV02DQ.html>

[9] Freescale Semiconductor, *MMA7261Q Datasheet.* Rev 1 2006-5 <http://inst.eecs.berkeley.edu/~ee100/fa08/lab/project/MMA7260Q-Rev1.pdf >

[10] SOUKUP J., *Distribuovaný varovný systém pro automobilovou bezpečnost,*Diplomová práce, Praha: České vysoké učení technické, katedra řídicí techniky,2007

[11] TinyOS, *Installing a XubunTOS Virtual Machine on VMWare Server* [online]. Poslední revize 2009-01-19 [cit.2009-04 -20]. <http://docs.tinyos.net/index.php/Running_a_XubunTOS_Virtual_Machine_Image_in_VMware_Player>

[12] Core Working Group, *Analog to Digital Converters (ADCs)* [online]. Poslední revize 2007-08-14 [cit.2009-04 -22]. <http://www.tinyos.net/tinyos-2.1.0/doc/html/tep101.html>

[13] Arch Rock Corporation*, MSP430 USART*[online]. Poslední revize není udána [cit.2009-04 -23]. <http://smote.cs.berkeley.edu:8000/tracenv/browser/HydroWatch/tinyos-2.x-quanto2/tos/chips/msp430/usart/>

[14] WIKIPEDIA, *TinyOS*[online].. Poslední revize není udána [cit.2009-04 -23]. <http://en.wikipedia.org/wiki/TinyOS >

[15] TinyOS, *Mote-mote radio communication* [online]. Poslední revize 2009-01-10 [cit.2009-04 -23]. <http://docs.tinyos.net/index.php/Mote-mote_radio_communication>

[16] Core Working Group, *message_t* [online]. Poslední revize 2007-08-14 [cit.2009-04 -23] <http://www.tinyos.net/tinyos-2.1.0/doc/html/tep111.html>
[17] Core Working Group, *Seriál Communication* [online]. Poslední revize 2008-04-07 [cit.2009-04 -23] <http://www.tinyos.net/tinyos-2.x/doc/html/tep113.html>
[18] tinyos-help, *interface Timer<TMicro>* [online]. Poslední revize 2007-05-10 [cit.2009-04 -23] < http://www.mail-archive.com/tinyos-help@millennium.berkeley.edu/msg12078.html>
[19] Core Working Group, *Timers* [online]. Poslední revize 2007-09-5 [cit.2009-04 -23] <http://www.tinyos.net/tinyos-2.x/doc/html/tep102.html>
[20] Core Working Group, *Resource Arbitration* [online]. Poslední revize 2007-08-14 [cit.2009-04 -23] <http://www.tinyos.net/tinyos-2.1.0/doc/html/tep108.html>
[21] *Úvod do kompresních algoritmů* [online]. Poslední revize není udáno [cit.2009-04 -23] <*https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7019*>
[22] *Compression* [online]. Poslední revize není udáno [cit.2009-04 -23] <http://www.cs.otago.ac.nz/cosc463/2005/compress.htm>

10. Přílohy

Seznam příloh:

- Příloha A – Složení a velikost zpráv v komunikačním protokolu
- Příloha B – Diagramy popisující propojení komponentů

Příloha A

Rozdělení zpráv podle id zprávy

F1 – Inicializační zpráva

Velikost: 2 B
Složení:

- Id zprávy – 1 B
- Id jednotky master – 1 B

F2 – Nastavovací zpráva

Velikost: 13 B
Složení:

- Id zprávy – 1 B
- Id master – 1 B
- Verze nastavovací zprávy – 2 B
- Vzorkovací frekvence akcelometru – 1 B
- Číslo smyčky žádosti o naměřená data – 2 B
- Metoda potvrzovaní – 1 B
- Vysílací kanál – 1 B
- Lokální čas jednotky master – 4 B

F3 – Identifikační zpráva

Velikost: 2 B
Složení:

- Id zprávy – 1 B
- Id slave - 1 B

F4 – Žádost o naměřená data

Velikost: 6 B
Složení:

- Id zprávy – 1 B
- Id master – 1 B
- Verze nastavovací zprávy – 2 B
- Číslo smyčky žádosti o naměřená data – 2 B

F5 – Žádost o nastavovací zprávu

Velikost: 4 B
Složení:

- Id zprávy – 1 B
- Id slave – 1 B
- Verze nastavovací zprávy – 2 B

F6 – Nekomprimovaná zpráva s naměřenými daty

Velikost: 25 B
Složení:

- Id zprávy – 1 B
- Id slave – 1 B
- Číslo smyčky žádosti o naměřená data – 2 B
- Verze nastavovací zprávy – 2 B
- Celkový počet datových rámců – 1 B
- Číslo datového rámce – 1 B
- Vzorkovací frekvence akcelometru – 1 B
- Data se vzorky měření bez komprese – 16 B

F7 – Potvrzovací zpráva

Velikost: 4 B
Složení:

- Id zprávy – 1 B
- Id master – 1 B
- Celkový počet datových rámců – 1 B
- Číslo datového rámce – 1 B

F8 – Zpráva o odeslání všech datových rámců

Velikost: 6 B
Složení:

- Id zprávy – 1 B
- Id slave – 1 B
- Číslo smyčky žádosti o naměřená data – 2 B
- Celkový počet datových rámců – 1 B
- Číslo posledního vzorku měření – 1 B

F9 – Zpráva s čísly nedoručených datových rámců

Velikost: 24 B
Složení:

- Id zprávy – 1 B
- Id master – 1 B
- Celkový počet datových rámců – 1 B
- Číslo datového rámce – 1 B
- Čísla nepřijatých datových rámců – 20 B

F10 – Komprimovaná zpráva s naměřenými daty

Velikost: 24

- Složení: Id zprávy – 1 B
- Id slave – 1 B
- Celkový počet datových rámců – 1 B

- Číslo posledního vzorku měření z datového rámce – 1 B
- Pomocné hodnoty pro zjištění, jestli jsou rozdíly kladné nebo záporné – 4 B
- Data se vzorky měření s komprese – 16 B

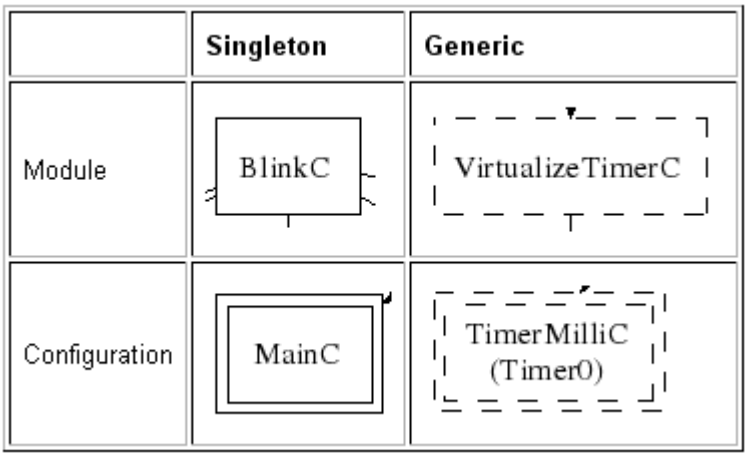
F11 – Zpráva k uspání jednotky slave

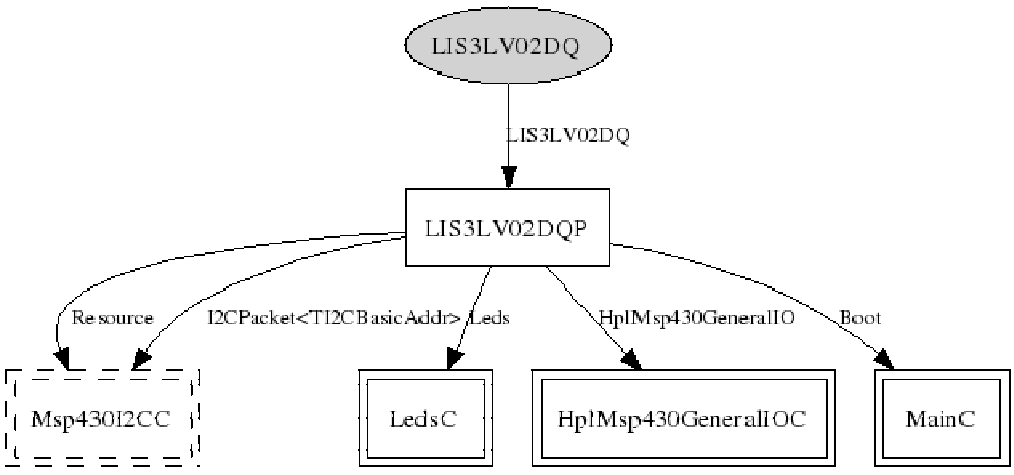
Velikost: 6 B
Složení:

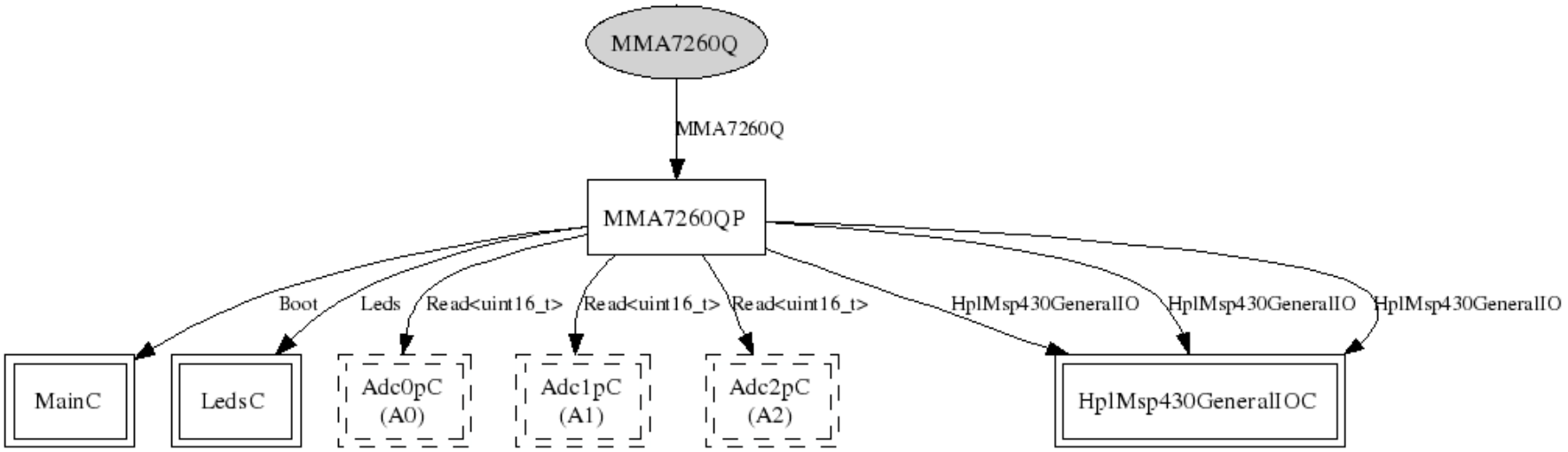
- Id zprávy – 1 B
- Id master – 1 B
- Doba, po kterou má být jednotka slave uspaná – 4 B

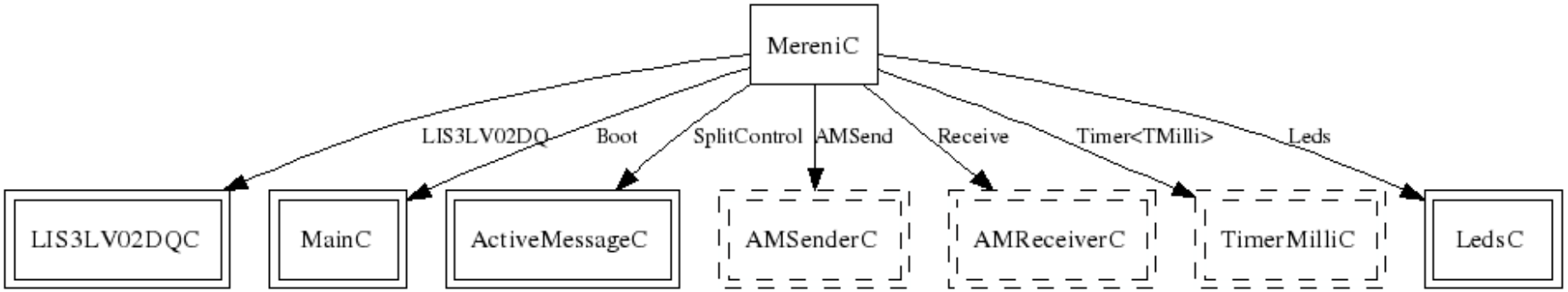
Příloha B

Obr. 1 – Označení v diagramech Module a Configuration
Obr. 2 – Diagram propojení Module LIS3LV02DQ s ostatními komponenty pomoci Configuration
Obr. 3 – Diagran propojení Module MMA7260Q s ostatními komponenty pomoci Configuration
Obr. 4 – Diagram propojení Module MereniC (sw v jednotce slave) s ostatními komponenty pomoci Configuration
Obr. 5 – Diagran propojení Module BaseStation (sw v jednotce master) s ostatními komponenty pomoci Configuration






MMA7260Q
MMA7260Q
MMA7260QP
Boot
Leds
Read<uint16_t>
Read<uint16_t>
Read<uint16_t>
HplMsp430GeneralIO
HplMsp430GeneralIO
HplMsp430GeneralIO
MainC
LedsC
Adc0pC
(A0)
Adc1pC
(A1)
Adc2pC
(A2)
HplMsp430GeneralIOC


MereniC
LIS3LV02DQ
Boot
SplitControl
AMSend
Receive
Timer<TMilli>
Leds
LIS3LV02DQC
MainC
ActiveMessageC
AMSenderC
AMReceiverC
TimerMilliC
LedsC

MainC
ActiveMessageC
SerialActiveMessageC
LedsC
BaseStationP
BaseStation
TimerMilliC
(Timer)
TimerMilliC
(TimerPhoto)
TimerMilliC
(TimerInc)
TimerMilliC
(TimerSyn)
TimerMilliC
(TimerStep)
CounterToLocalTimeC
LocalTime<TMilli>
UserButtonC
CC2420ControlC
CC2420SpiC
(Spi)
CounterMilli32C
Counter<TMilli, uint32_t>
Boot
Receive[am_id_t]
Packet
AMPacket
SplitControl
AMSend[am_id_t]
Receive[am_id_t].SplitControl
AMSend[am_id_t]
Receive[am_id_t]
Packet
AMPacket
Leds
Timer<TMilli>
Timer<TMilli>
Timer<TMilli>
Timer<TMilli>
Timer<TMilli>
Get<button_state_t>
Notify<button_state_t>
CC2420Config
CC2420Packet
CC2420Packet