UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Tracking system

Jan Ropek

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	[illegible]
Osobní číslo:	[illegible]
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	[illegible]
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

[illegible]

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***Kathy Sierra, Bert Bates: Head First Java, O'Reilly Media 2005, počet stran 720, ISBN-10 1600330002**

***Kline Kevin, Kline Daniel, Hunt Brand: SQL in a Nutshell, O'Reilly Media 2008, počet stran 592, ISBN-10 0596518846**

***Ajit Sagar, Sue Spielman a kol.: Professional Java Server Programming J2Ee 1.4 Edition, Wrox Press 2003, počet stran 850, ISBN-10 1861008139**

***Vivek Chopra, Sing Li, Jeff Genender: Professional Apache Tomcat 6, WROX Press 2006, ISBN-10: 0471753610**

***Sun Microsystems: The Java EE5 Tutorial [online], 2007 [cit. 2009-10-08],dostupný z: http://java.sun.com/javaee/5/docs/tutorial/doc/docinfo.html**

Vedoucí bakalářské práce: [redacted]
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. [redacted], Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne Jan Ropek

Poděkování

Na tomto místě chci poděkovat vedoucímu práce Ing. Pavlu Rozsívalovi za poskytnuté odborné rady a připomínky k této práci a celkově pohotovému přístupu k řešení problémů. Dále bych chtěl velmi poděkovat Ing. Jiřímu Krskovi za SW i HW podporu a odbornou konzultaci, včetně morální i technické pomoci během celého roku, kdy práce byla tvořena.

Anotace

Teoretická část pojednává o různých metodách určování polohy, včetně jednoho, který byl použit, metody přenosu dat mezi Master a Slave zařízení a základní představení použitých mikrokontrolérů, včetně všech obvodů, jako je například elektronický kompas.Následující kapitola se zabývá vývojovým prostředím. Třetí kapitola je věnována praktické části, kde jsou ukázány použité vývojové kity, z kterých je zařízení sestaveno. V závěru je shrnutí výsledků práce a praktická ukázka funkčnosti.

Klíčová slova

GPS, RF, ARM, AVR, Lora™

Title

Tracking system

Annotation

The theoretical part discusses the various methods of determining positions, including one that was used, methods of transferring data between the Master and Slave devices and basic introduction to microcontrollers used including all the circuits such as an electronic compass. The following chapter deals with the development environment. The next chapter shows the practical part of the particular modules and kits from which the product is constructed. Conclusion shows practical demonstration.

Keywords

GPS, RF, ARM, AVR, Lora™

Obsah

Seznam zkratek

GPS	Global Positioning System
ISM	Industrial, Scientific and Medical
GSM	Global System for Mobile Communications
TA	Timing Advance
BTS	Base transceiver station
RF	Radio frequency
TDOA	Time Difference of Arrival
NMEA	National Marine Electronics Association
LoRa	Long Range
OOK	On-Off Keying
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift Keying
MSK	Minimum Shift Kezing
GMSK	Gaussian Frequency Shift Keying
AGC	Automatic Gain Control
AFC	Automatic Frequency Control
CRC	Cyclic Redundancy Check
BW	Band Width
SF	Spreading Factor
FIFO	First In, First Out
SPI	Serial Peripheral Interface
DPS	Deska Plošných Spojů
RISC	Reduced Instruction Set Computing
ALU	Arithmetic Logic Unit
MCU	Microcontroller Unit

EEPROM	Electrically Erasable Programmable Read-OnlyMemory
SRAM	Static Random Access Memory
TWI	Two-wire Serial Interface
USART	Universal Synchronous Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
CPU	Central Processing Unit
D/A	Digital to Analog Converter
USB	Universal Serial Bus
TCB	Task Control Board
RAM	Random Acces Memory
LCD	Liquid Crystal Display
ASF	Atmel Software Framework
TFT	Thin Film Transtistor

Seznam obrázků

1 Úvod

Tato práce se zabývá sestrojením sledovacího zařízení, které k určení pozice využívá systém GPS. Cílem je sestrojit výrobek, který může na vzdálenost až 5 km (v závislosti na podmínkách terénu – zástavba, volný prostor) sledovat pozici určitého objektu.

Uživatel, který bude chtít pozorovat vybraný objekt, bude mít v ruce Master zařízení, které mu bude na displeji ukazovat, jak daleko se objekt nachází a jakým směrem se za ním vydat. Pokud například budeme chtít sledovat psa, musíme k němu přidělat Slave zařízení, což se provede například jeho zabudováním do obojku.

Komunikace mezi Master – Slave zařízeními probíhá v bezplatném ISM pásmu na frekvenci 869.525 MHz, které je dnes už velmi zahlcené a proto spolehlivost přenosu velmi závisí na prostředí, kde se přenos odehrává.

Uplatnění přístroje bude v budoucnu velmi široké. Pro sledování loveckého psa, či zabránění ztrátě Vašeho dítěte v rušném zábavném parku. V budoucnu se také implementuje GSM přenos, čímž velmi vzroste oblast použití, např. sledování pohybu služebních aut (dosah přístroje bude teoreticky neomezený).

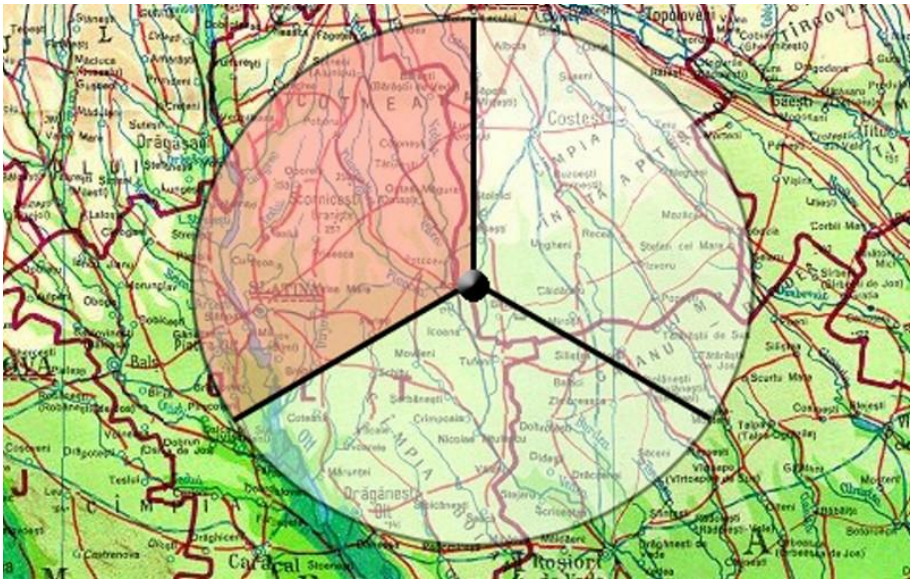
2 Teoretická část

2.1 Určení pozice

K získání informace o poloze se dají využít různé systémy. K nejvíce vhodným možnostem pro daný projekt patří systém GPS a GSM. V práci je prozatím použit pouze systém GPS.

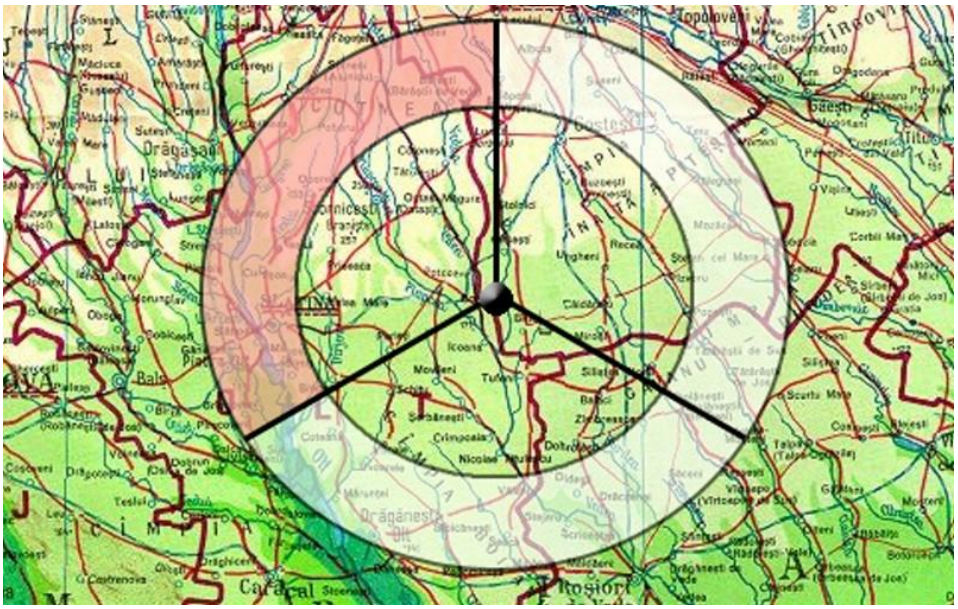
2.1.1 Určení polohy pomocí GSM

Jako nejjednodušší, ale zároveň nejméně přesný způsob zaměření mobilního zařízení je pomocí základnové stanice, se kterou mobil komunikuje. Jelikož je poloha základen známa, dá se díky tomu poloha stanovit. Ovšem nevýhodou je velká nepřesnost, neboť poloměr buněk může být třeba až 20 km. Každá buňka si však rozděluje vysílací prostor na několik sektorů – tím lze údaj o poloze trochu zlepšit, ale pro naše účely to ani tak není dostačující.



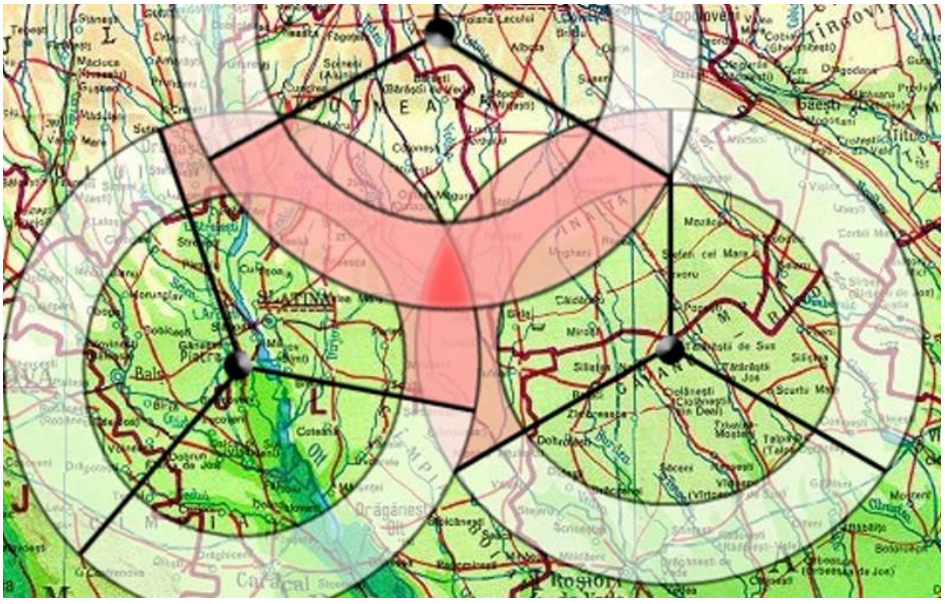
Obrázek 1 – Kruhová výseč [1]

Trochu přesnější metoda využívá předešlého systému a je doplněn o funkci TA (Timing Advance), která bere v úvahu zpoždění signálu mezi mobilní a základnovou stanicí. Z rozdílu časů, které vznikají zpožděním signálu, se dá poloha v dané kruhové výseči zpřesnit na mezikruží v rozmezí 1 km.



Obrázek 2 – TA [1]

Jako nejpřesnější určení se dnes provádí pomocí triangulace. Prakticky se jedná o zaměření mobilní stanice pomocí několika okolních základnových stanic BTS také s využitím TA. Hledá se vždy průsečík kružnic, které určí relativně přesně pozici. Princip triangulace s použitím TA je uveden na obrázku níže.



Obrázek 3 – Triangulace [1]

2.1.2 Určení polohy pomocí GPS [1]

GPS systém je vojenský globální družicový polohový systém, který provozuje Ministerstvo obrany Spojených států. Umožňuje relativně přesně určit přesnou polohu na zemi (řádově jednotky metrů), což je pro tuto aplikaci dostačující.

Na scéně se začínají objevovat nové projekty jako GALILEO (evropský zástupce), nebo GLONASS (zástupce Ruska). Tyto služby však zatím nebudou v práci použity.

GPS systém má na oběžných drahách země 32 družic, které obíhají ve výšce 20 350 km nad povrchem Země na 6ti kruhových drahách. Váha jedné družice je cca 1.8 tuny. Doba oběhu 11 hodin a 58 minut. Srdcem systému jsou atomové hodiny s přesností 10^{-13} s. Každá družice vysílá signál (zprávu) o své poloze a přibližné poloze ostatních družic. Čím více družic přijímač dokáže zachytit, tím přesněji se dá určit poloha.

Přijímač GPS je tvořen anténou, RF jednotkou, mikroprocesorem a zdrojem napětí. GPS zpracovává signál využitím korelace. Přijímač porovnává PRN kód s přijímaným signálem a hledá se stav, kdy korelace je rovna jedné. V tom okamžiku je znám jak čas družicových hodin, tak aktuální čas v přijímači. Signál vyslaný s družice v sobě nese údaj o čase (v kterém byl signál odeslán), který se porovná s časem v GPS přijímači a pouhým rozdílem těchto časů se dá určit vzdálenost od družice k objektu (pokud zanedbáme Einsteinovu Obecnou teorii relativity). Pokud máme signály z alespoň 4 satelitů, dokážeme určit poziční bod.

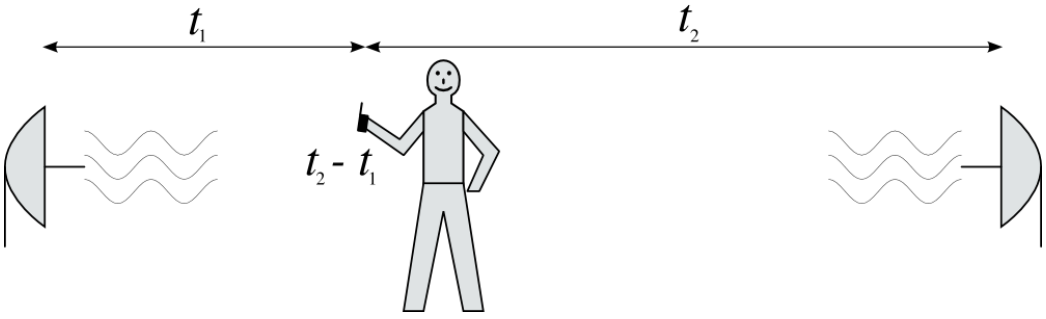
Družice jsou nastaveny tak, že v navzájem synchronizovaném okamžiku společně vyšlou signál. Přijímač pak svoji polohu spočítá pomocí časových rozdílů, v kterých jednotlivé signály přijmul. Tento systém se jmenuje TDOA (*Time Difference of Arrival*).

Pro stanovení pozice na přímce platí:

Časový rozdíl ze dvou přijatých signálů je roven:

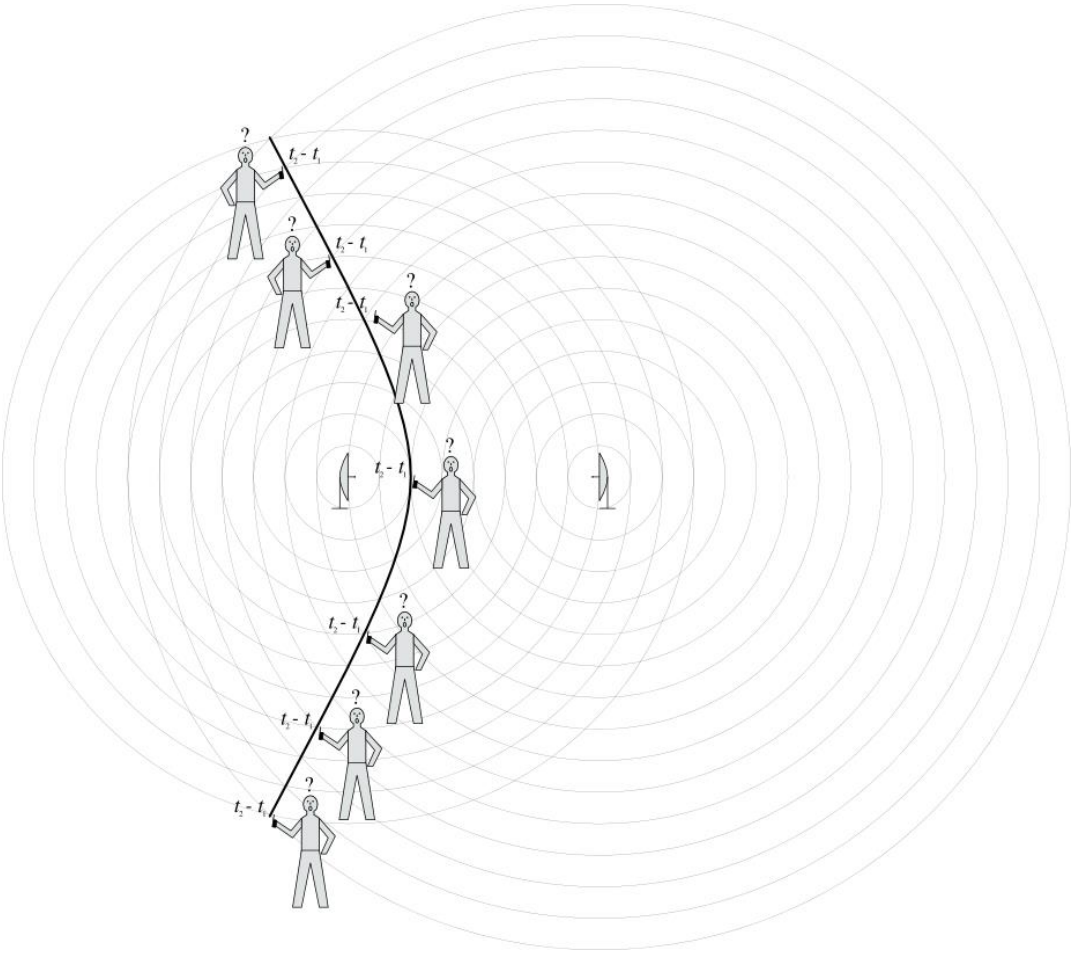
$$\Delta t = t_2 - t_1 \qquad (2)$$

Pokud známe rychlost šíření signálu cca 300 000 km/s, lze snadno dopočítat vzdálenost přijímače od jednotlivých vysílačů.



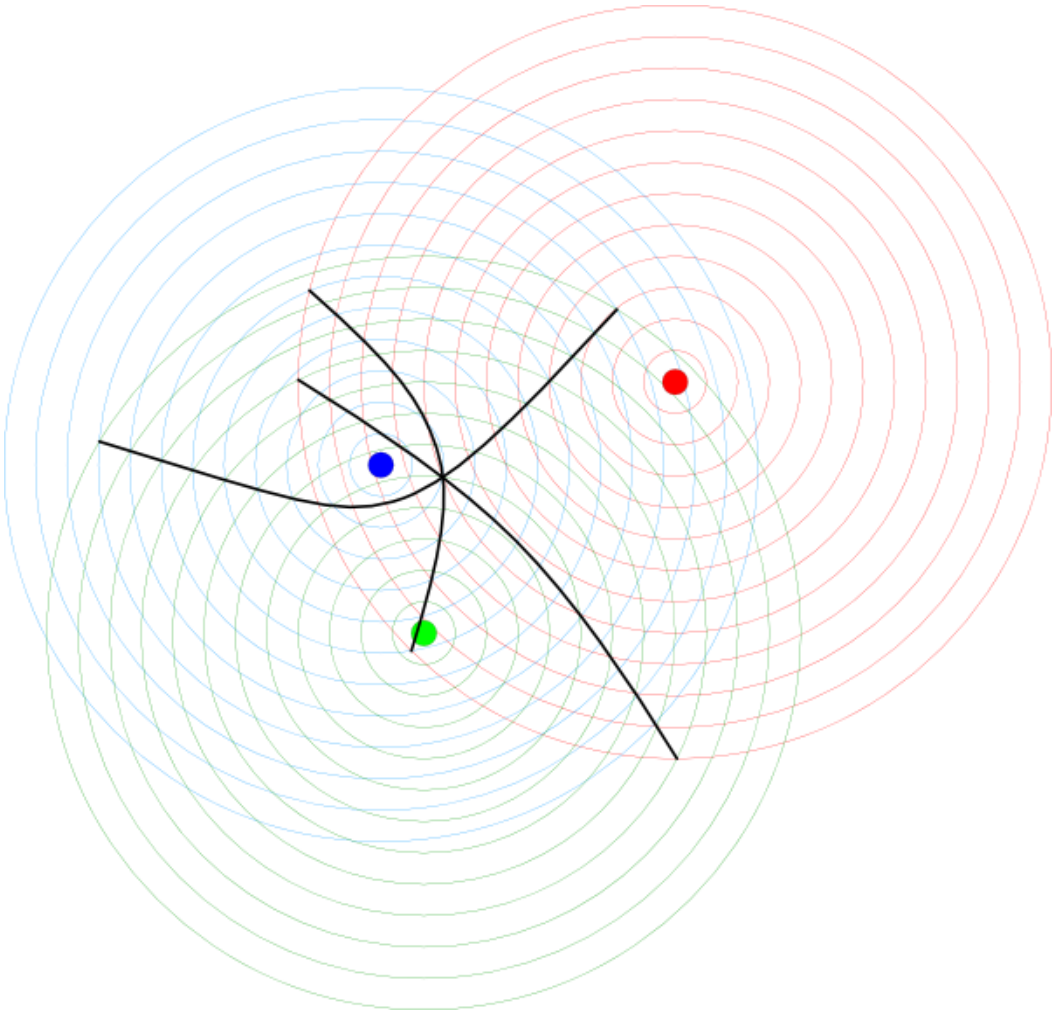
Obrázek 4 – Stanovení pozice na přímce [1]

Pokud chceme určit pozici v rovině, je zapotřebí přijmout signál nejméně ze tří satelitů. Pokud bychom měli signál pouze ze dvou satelitů, poloha by se nedala přesně určit. Můžeme jen konstatovat, že se přijímač nachází někde na hyperbole vysílače (hyperbola má totiž tu vlastnost, že všechny body, které na ní leží, mají stejný rozdíl vzdáleností od obou ohnisek – kde se nachází satelity). Určení polohy v rovině pouze ze dvou satelitů vidíme na obrázku níže:



Obrázek 5 – Stanovení pozice v rovině [1]

Pokud máme vysílače nejméně tři, získáme přesný průsečík, který určí polohu.



Obrázek 6 – Stanovení pozice v rovině [1]

K určení polohy v prostoru jsou zapotřebí vysílače čtyři. Tři vysílače nám určí zeměpisnou šířku i délku, ale neposkytne nám znalost nadmořské výšky. Pokud tedy máme signál pouze ze tří satelitů, je jako nadmořská výška brána pouze obecná (průměrná) nadmořská výška a výsledkem je tedy pouze odhad souřadnic, kde navíc údaj o nadmořské výšce zcela chybí.

2.2 GPS modul

Byl vybrán modul od firmy Quectel L80, který disponuje malými rozměry, nízkou spotřebou (20 mA ve sledovacím módu) a především integrovanou anténou. Maximální citlivost je až -165 dBm.

Obrázek 7 – Gps modul – Quactel L80 [2]

2.2.1 Komunikace s modulem (interface)

L80 má k dispozici pouze sériovou sběrnici UART s přenosovou rychlostí od 4800 do 115200 bps. Defaultní hodnota je nastvena na 9600 bps. Modul se chová jako Multi-Master, tedy posílá v daných časových intervalech data do procesoru. Napájecí napětí se může pohybovat od cca 3 V do 4.3 V. Proto musel být vybrán mikroprocesor s podobným napájecím napětím, pro společnou obvodovou napájecí část. V tomto případě byla zvolena Xmega32E5 od firmy Atmel s udávaným Vcc od 1.6 V do 3.6 V.

2.2.2 Komunikační protokol

Mezi GPS se stal komunikačním standardem NMEA protokol 0183. Přijímač L80 tento protokol využívá v defaultním nastavení, umí ovšem i rozšířený MTK NMEA paketový protokol. Díky tomuto protokolu se dá v GPS přenastavit například rychlost komunikace nebo odesílané NMEA věty do Slave procesoru.

Filed	Length(bytes)	Description
$	1	Each NMEA message starts with '$'
Talker ID	1~2	'GP' for a GPS receiver.
NMEA message ID	3	NMEA message ID
Data Field	Variable, depend on the NMEA message type	Data fields, delimited by comma ','
*	1	End character of data field
Checksum	2	A hexadecimal number calculated by exclusive OR of all characters between '$' and '*'
<CR><LF>	2	Each NMEA message ends with 'CR' and 'LF'

Obrázek 8 – NMEA – struktura protokolu [2]

NMEA protokol disponuje těmito větami:

- GPRMC
- GPVTG
- GPGGA
- GPGSA
- GPGSV
- GPGLL
- GPTXT

2.2.3 NMEA věty

2.2.3.1 GPRMC

Obsahuje minimální doporučené informace pro navigaci, jako je pozice (aktuální souřadnice), čas v UTC, datum, aktuální rychlost v uzlech, kurz pohybu a magnetickou deklinaci – tedy úhlový rozdíl mezi zeměpisným a magnetickým severním pólem, přičemž se tento úhel posouvá asi o 6 úhlových minut k východu ročně.

2.2.3.2 GPVTG

Zahrnuje aktuální rychlost v uzlech i km/h včetně aktuálního směru pohybu.

2.2.3.3 GPGGA

Nese informaci o času, ve kterém byla získána poloha, souřadnice polohy, počet použitých družic k výpočtu, výšku nad elipsoidem WGS84, a kvalitu signálu.

2.2.3.4 GPGSA

Věta GPGSA obsahuje informaci o počtu viditelných družic a jejich geometrickém rozmístění (HDOP, VDOP, PHOP).

2.2.3.5 GPGSV

Obsahuje údaje o počtu viditelných družic a informace o každé z nich. Patří sem ID satelitu (PRN), elevace a azimut ve stupních a poměr signál šum SNR v dBHz tj. SNR +BW, kde BW je šířka pásma měření.

2.2.3.6 GPGLL

Obsahuje geografickou polohu a čas v UTC.

2.2.3.7 GPTXT

Obsahuje informace o výstupních větách, např. počet odesílaných vět.

2.3 Bezdrátový přenos

Jedna z možností, jak přenášet data mezi Master a Slave zařízením, je využití GSM. Značnou výhodou je neomezený dosah (v závislosti na pokrytí operátorů). Ovšem jako velká nevýhoda je cena za poskytnutí služeb pro přenos dat. Pro moji aplikaci tento druh komunikace zatím nebude použit.

Pro bezplatné získání bezdrátového přenosu slouží ISM pásmo. Pro tuto možnost existuje spousty RF modulů. Většina výrobců zajišťuje s vysílacím výkonem 20dBm návaznost spojení na vzdálenost maximálně 2 km za ideálních podmínek. To, ale pro dané zařízení není dostačující. Na trhu se ovšem vyskytla novinka (2013) od firmy Semtech, která slibuje dosah až na vzdálenost 15 km a to díky modulaci pracující na bázi rozprostřeného spektra. Určený bit k odeslání se neodešle na jedné frekvenci, jako je tomu u FSK modulace, ale každý vyslaný bit se rozprostře na několik frekvencí vyskytujících se velmi blízko sebe. Vybraný modul umožňuje jeden bit odeslat až na 4096 frekvencích. Přijímač se pak snaží s předem danou posloupností hledat jednotlivé špičky v signálu a má oproti FSK značnou výhodu, že pokud přijímač nenaleze signál na jedné frekvenci, tak má možnost ho najít na 4095 dalších frekvencí. Přesný princip není znám, neboť si jej firma Semtech nechala patentovat. Vše je pod registrovanou značkou LoRa™.

Jako nejvhodnější modul od firmy Semtech byl zvolen obvod SX1276, který zaručuje citlivost přijímaného signálu až na úroveň -146.5 dBm s maximální rychlostí přenosu dat 300 kbps. Modul disponuje všemi základními modulacemi jako OOK, FSK, GFSK, MSK, GMSK a nestandardní LoRa™ modulací, díky které jde získat data ze signálu o úrovni až zmiňovaných -146.5 dBm, ovšem jen při velmi nízkých přenosových rychlostech.

2.3.1 Modul SX1276

Jelikož použitý mikrokontrolér, GPS modul i displej pracují na 3 V logice, je velmi přínosné, že SX1276 také vystačí s napájecím napětím kolem 3 V. Maximální dovolené napětí je pak 3.9 V a minimální 1.8 V.

Za zmínku stojí také velmi nízká spotřeba v přijímacím režimu, která činí 12 mA. Při vysílacím režimu s maximálním výstupním výkonem je pak odběr cca 120 mA. SX1276 disponuje funkcemi jako AGC či AFC. AFC slouží k automatické korekci frekvence pro komunikaci (které nelze použít pro Lora modulaci) a AGC automaticky řídí zisk na přijímači a přizpůsobuje tak spotřebu celého modulu.

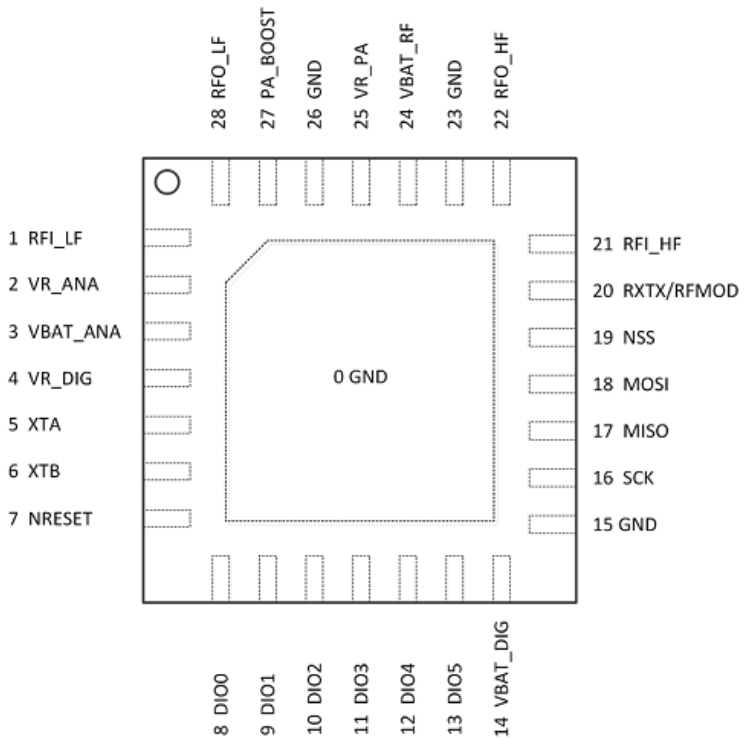
Modul má několik režimů, spotřeba každého z nich je uvedena v tabulce.

Symbol	Description	Conditions	Min	Typ	Max	Unit
IDDSL	Supply current in Sleep mode		-	0.2	1	uA
IDDIDLE	Supply current in Idle mode	RC oscillator enabled	-	1.5	-	uA
IDDST	Supply current in Standby mode	Crystal oscillator enabled	-	1.6	1.8	mA
IDDFS	Supply current in Synthesizer mode	FSRx	-	5.8	-	mA
IDDR	Supply current in Receive mode	*LnaBoost* Off, band 1 *LnaBoost* On, band 1 Bands 2&3	- - -	10.8 11.5 12.0	- - -	mA
IDDT	Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST RFOP = +17 dBm, on PA_BOOST RFOP = +13 dBm, on RFO_LF/HF pin RFOP = + 7 dBm, on RFO_LF/HF pin	- - - -	120 87 29 20	- - - -	mA mA mA mA

Obrázek 9 – SX1276 - režimy [3]

Klíčové vlastnosti modulu SX1276:

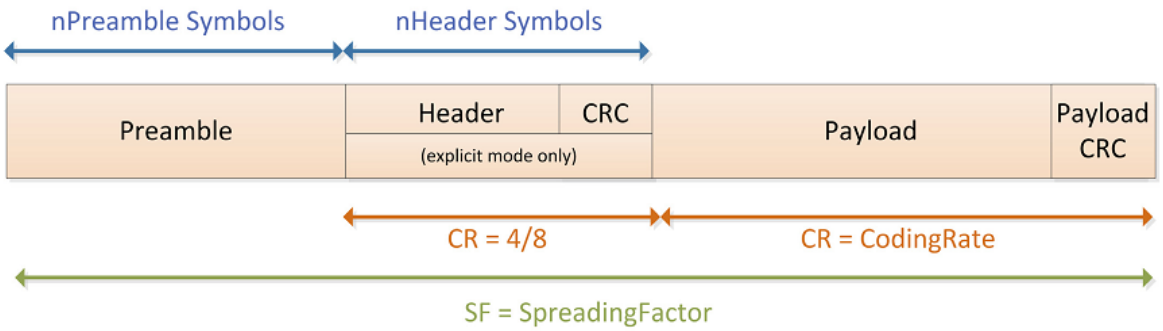
- + 20 dBm vysílací výkon
- Programovatelná přenosová rychlost až do 300 kbps
- Vysoká citlivost až -146.5 dBm
- Nízký odběr při zapnutém příjmu 10 mA, ve SLEEP režimu 200 nA
- Modulace FSK, GFSK, OOK, MSK, GMSK, LoRa™
- Preamble detekce
- Rozpoznání synchronizačního slova
- AFC + AGC
- Délka paketu až 64 B s CRC
- Teplotní sensor + detektor nízkého napětí



Obrázek 10 – Pouzdro a rozmístění vývodů [3]

2.3.1.1 Komunikační paket SX1276

Komunikační paket je složen z několika dílčích segmentů.



Obrázek 11 – Rozložení paketu SX1276 [3]

- **Preamble –** Slouží k synchronizování vysílače s přijímačem, jedná se o pravidelně se střídající sled nulových a jedničkových bitů 0xAA . Délka preamble se dá nastavit. Pro běžné účely se však používá 8 – 12 symbolů.
- **Header (Hlavička) –** Obsahuje informace jako je délka paketu, druh použitého CRC a může obsahovat i CRC složeného pouze z hlavičky.

- **Payload –** Je pole, do kterého se ukládájí data (informace) určená k přenosu (jedná se o FIFO). Payload také může být doplněn o CRC, který je automaticky spočítán modulem SX1276.

Crc Type	*CrcWhiteningType*	Polynomial
CCITT	0 (default)	$X^{16} + X^{12} + X^5 + 1$
IBM	1	$X^{16} + X^{15} + X^2 + 1$

Obrázek 12 – Rozložení paketu SX1276 [3]

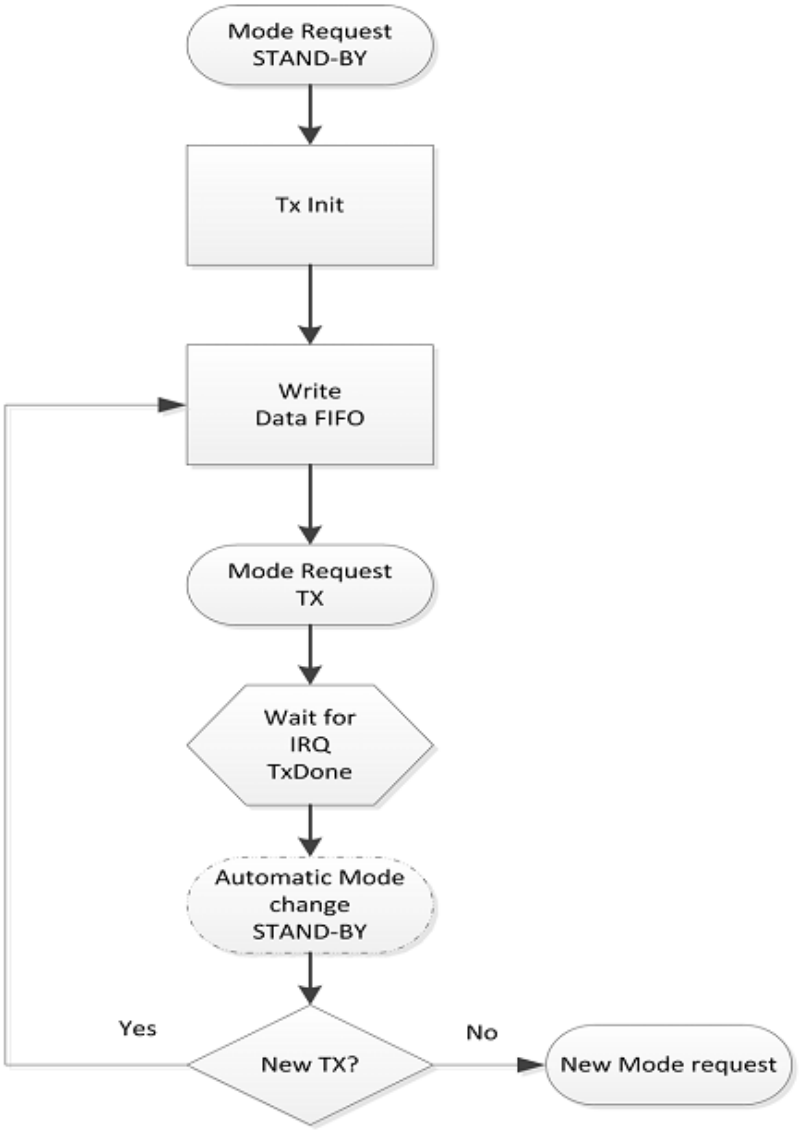
Přenosová rychlost – lze ji nastavit v rozmezí 18 až 300 000 bps, pro povahu aplikace nepotřebuji velkou rychlost přenosu, ale maximální možnou vzdálenost spojení. Proto jsem nastavil přesně 293 bps.

Vzorec: udává symbolovou rychlost

$$Rs = \frac{BW}{2^{SF}} \qquad (1)$$

Kde BW je šířka pásma a SF je parameter pro nastavení rozprostřeného spektra.

2.3.1.2 Sekvence odesílání dat

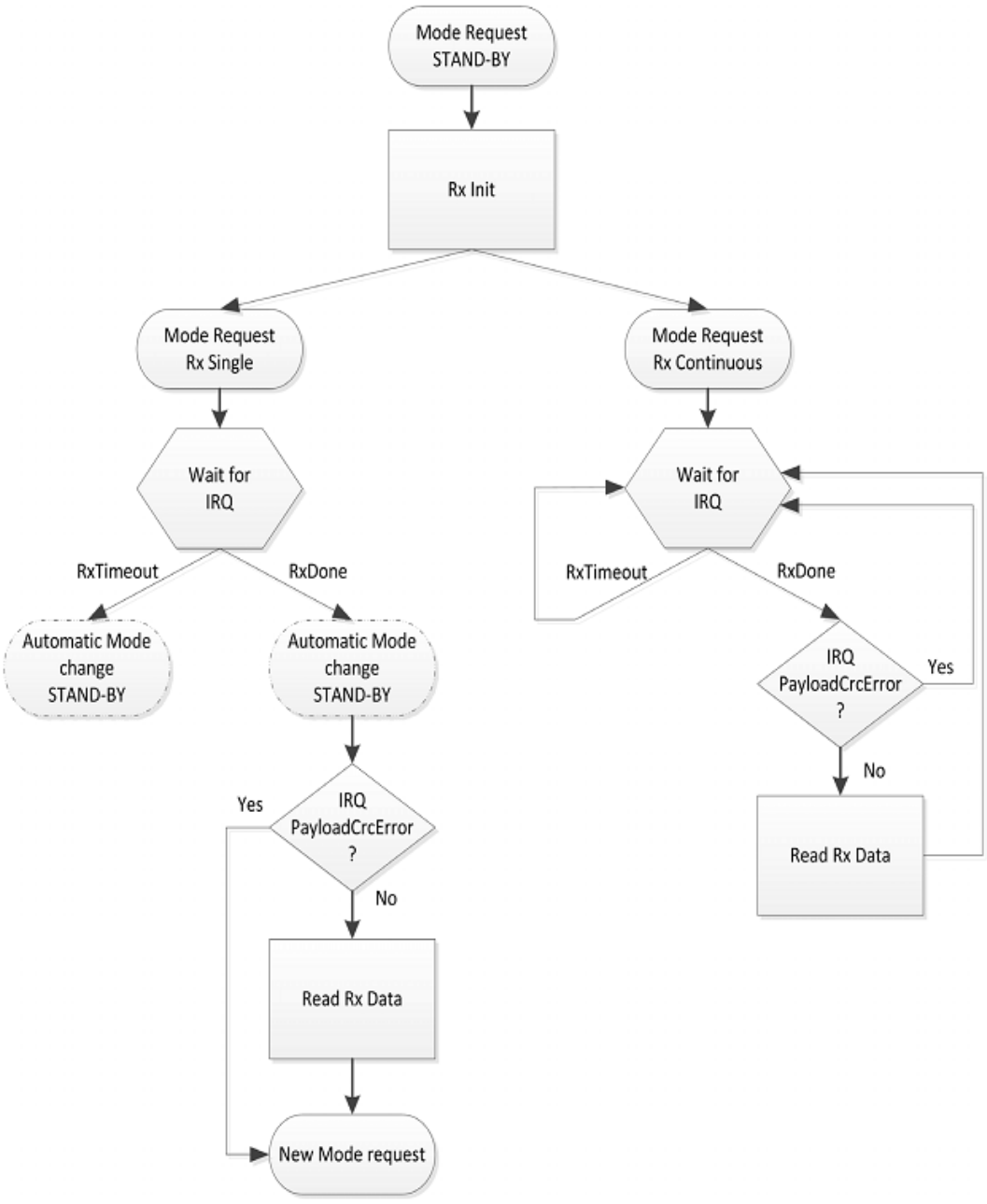


Obrázek 13 – Rozložení paketu SX1276 [3]

Z algoritmu jsou vidět jednotlivé kroky, které jsou potřeba udělat pro odeslání dat.

- Inicializace modulu
- Zapsat data do vysílací TX FIFO paměti
- Nastavit události, pro generaci přerušení na připojeném mikrokontroléru
- Počkat na přerušení TxDone – data odeslána
- Přejít do STAND-BY režimu
- Proces můžeme opakovat

2.3.1.3 Sekvence příjmu dat



Obrázek 14 – Rozložení paketu SX1276 [3]

Z algoritmu jsou vidět jednotlivé kroky, které jsou potřeba udělat pro příjem dat.

- Inicializace modulu
- Zvolit přijímací režim Single/Continuous
- Vyčkat na přerušení – data přijata nebo vypršení nastaveného časového limitu

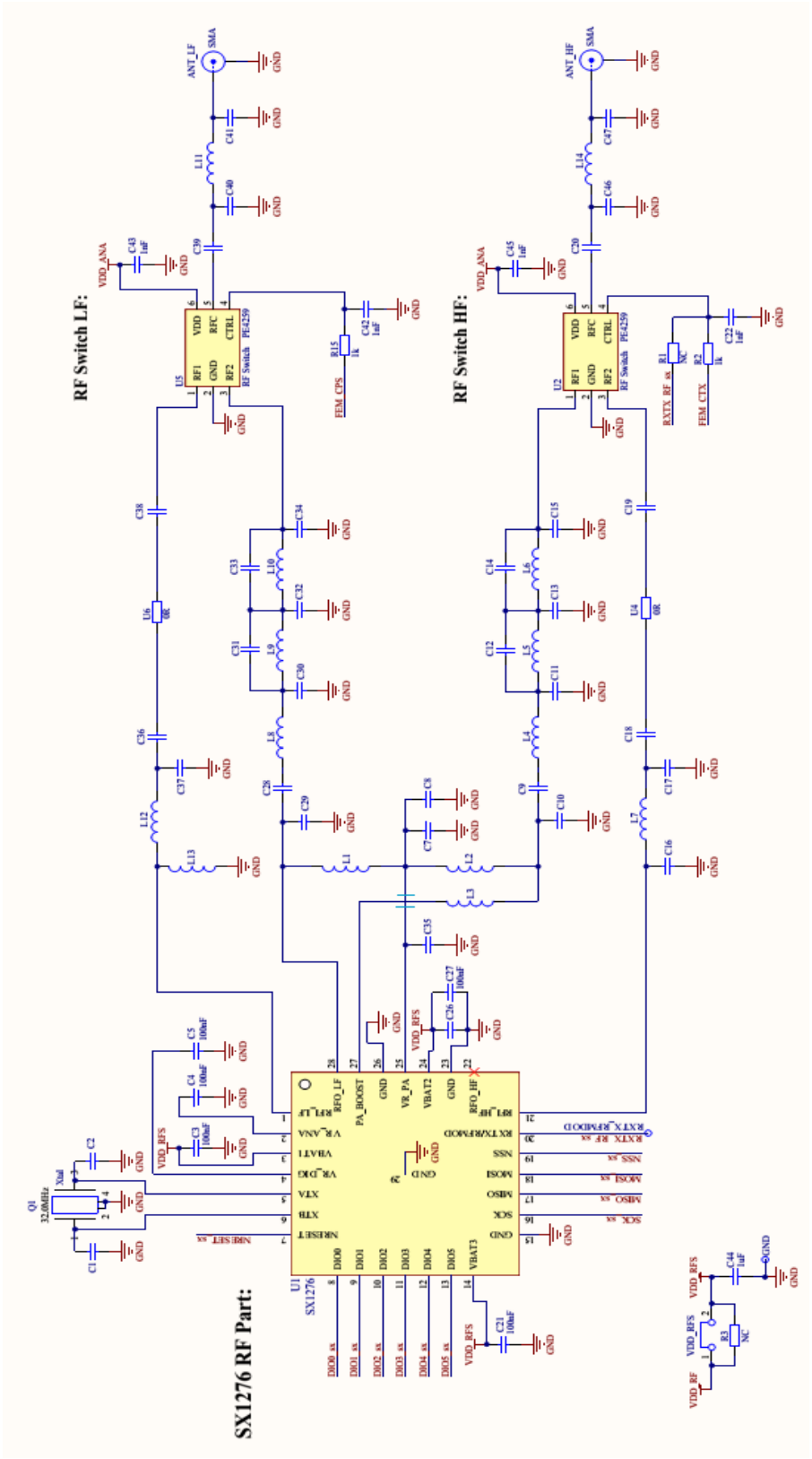
V případě Single režimu:

I přesto, že nám modul dal přerušením vědět o přijatých datech, neznamená to,že vše proběhlo správně. Je zde možnost, že paket je nějak poškozen nebo nedorazil celý, proto musíme vyčkat na další přerušení od modulu, kterým si zkontrolujeme, zda sedí vypočítaný CRC s přijmutým. Pokud se CRC neshoduje, data z RX FIFO nečteme (jsou poškozená), v opačném případě přijatá data můžeme použít. Poté můžeme celý příjem opakovat nebo nastavit jiný požadovaný režim.

V případě Continuous režimu:

Jedná se o obdobný postup, jen s tím rozdílem, že pokud CRC nesedí, automaticky modul přejde do nového příjmu. Jestliže je CRC v pořádku, přečtou se data z FIFO a pokračuje se opět v příjmu.

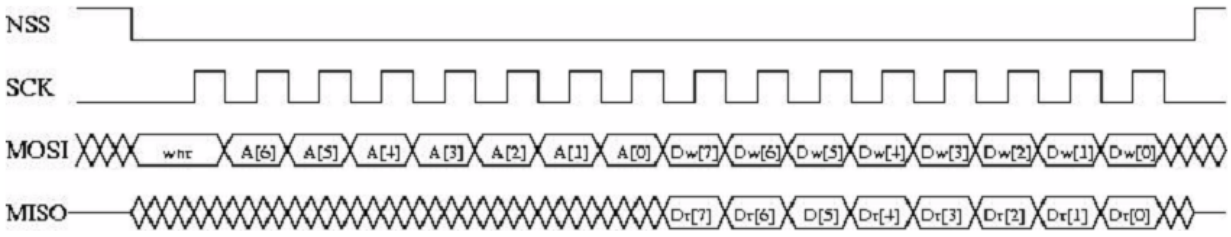
2.3.1.4 Schéma zapojení modulu



Obrázek 15 – Rozložení paketu SX1276 [3]

2.3.1.5 Komunikace s modulem SX1276

Nastavení všech registrů v modulu se provádíí skrze SPI sběrnici, kde mikrokontrolér se chová jako Master. Typický přenos dat je ukázán na následujícím obrázku.



Obrázek 16 – SPI komunkace [3]

Přenos vždy začíná výběrem daného zařízení, se kterým chceme komunikovat, tedy položení NSS signálu do logické nuly. Pak proběhne přenos dat, kde SCK jsou hodiny, MOSI jsou data vystupující z mikrokontroléru a MISO jsou čtené hodnoty odesílané Slave zařízením SX1276. Po skončení přenosu se opět NSS uvede do logické úrovně 1.

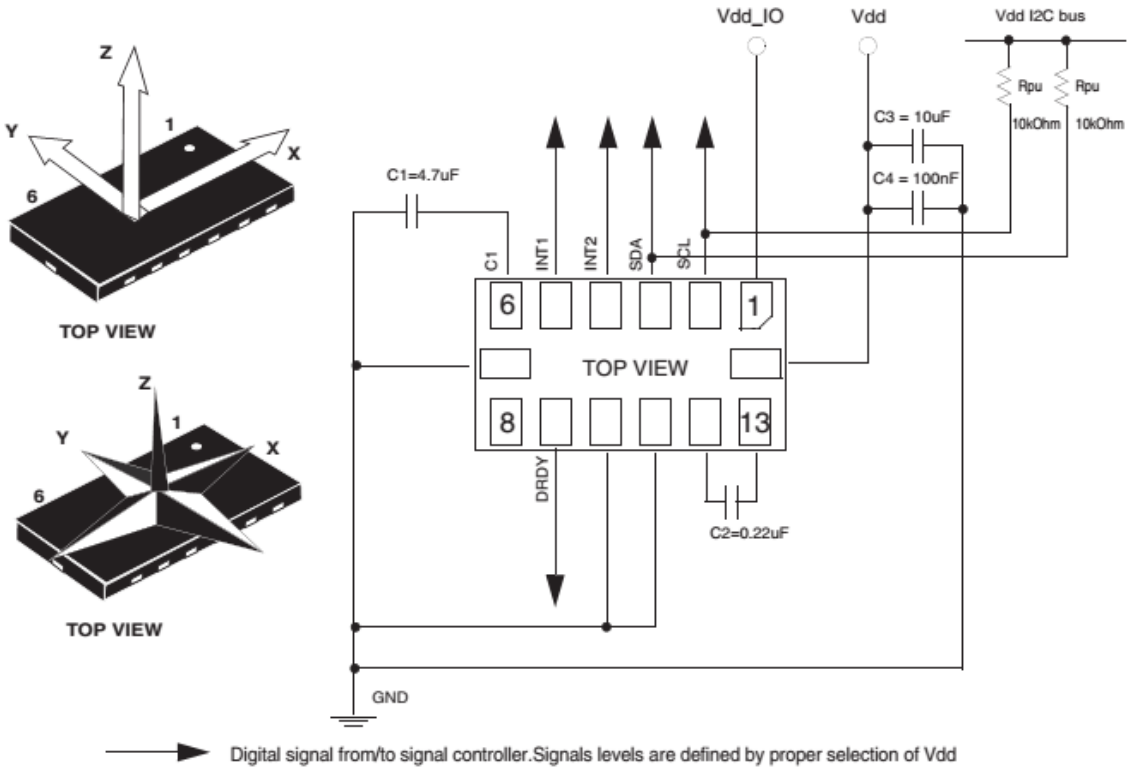
První vyslaný byte obsahuje adresu,na kterou chceme přistupovat (adres je celkem 128), kde posledním bitem sdělíme, zda z dané adresy budeme číst nebo do ní zapisovat. Dalším vyslaným bytem je pro případ zápisu hodnota, kterou chceme na adresu uložit a pro případ čtení je tato hodnota nepodstatná – tento byt se proto právem označuje *Dummy*.

Pokud chceme číst celý buffer (například FIFO registr), můžeme využít funkci, která umožňuje číst data kontinuelně. Master mikrokontrolér nejdříve NSS pinem vybere zařízení, pak vyšle adresu, z které chce začít číst celý buffer a pokud po přečtení první hodnoty zůstane NSS pin v nulové logické úrovni, při dalším opakovaném čtení se automaticky uvnitř SX1276 posune ukazatel na následující adresu. Tento postup lze aplikovat jak na čtení, tak na zápis. Výhoda je urychlení celého procesu komunikace mezi zařízeními.

2.4 Elektronický kompas

Pro určení směru k Slave zařízení je potřeba znát orientaci (úhel natočení) k určitému konstantnímu bodu – severní pól. Pro tuto potřebu jsem využil elektronického kompasu LSM303DLHC se zabudovaným akcelerometrem.

Veškerá komunikace probíhá přes TWI sběrnici s maximální možnou přenosovou rychlostí 400 KHz. Také je zde podpora přerušení – například, že nová data jsou připravena ke čtení.



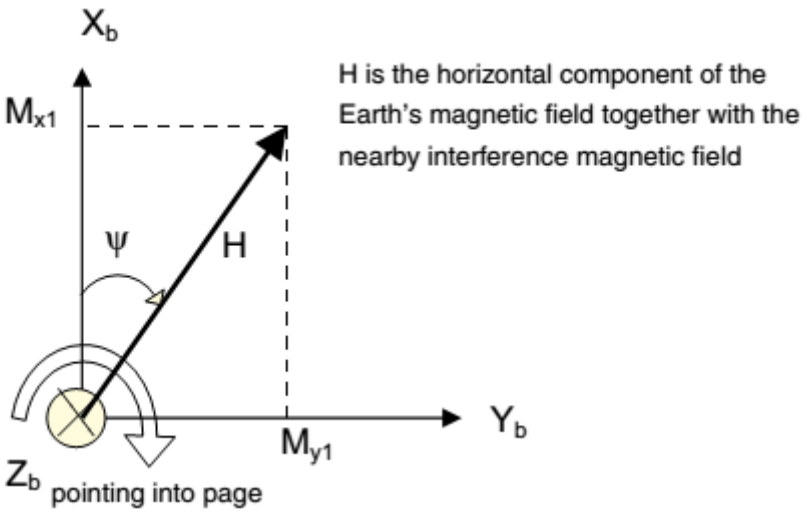
Obrázek 17 – Orientace os + zapojení [4]

Kompas umožňuje nastavení několik rychlostí převodu, viz. tabulka níže:

ODR3	ODR2	ODR1	ODR0	Power mode and ODR selection
0	0	0	0	Power-down mode
0	0	0	1	Normal / low-power mode (1 Hz)
0	0	1	0	Normal / low-power mode (10 Hz)
0	0	1	1	Normal / low-power mode (25 Hz)
0	1	0	0	Normal / low-power mode (50 Hz)
0	1	0	1	Normal / low-power mode (100 Hz)
0	1	1	0	Normal / low-power mode (200 Hz)
0	1	1	1	Normal / low-power mode (400 Hz)
1	0	0	0	Low-power mode (1.620 kHz)
1	0	0	1	Normal (1.344 kHz) / low-power mode (5.376 kHz)

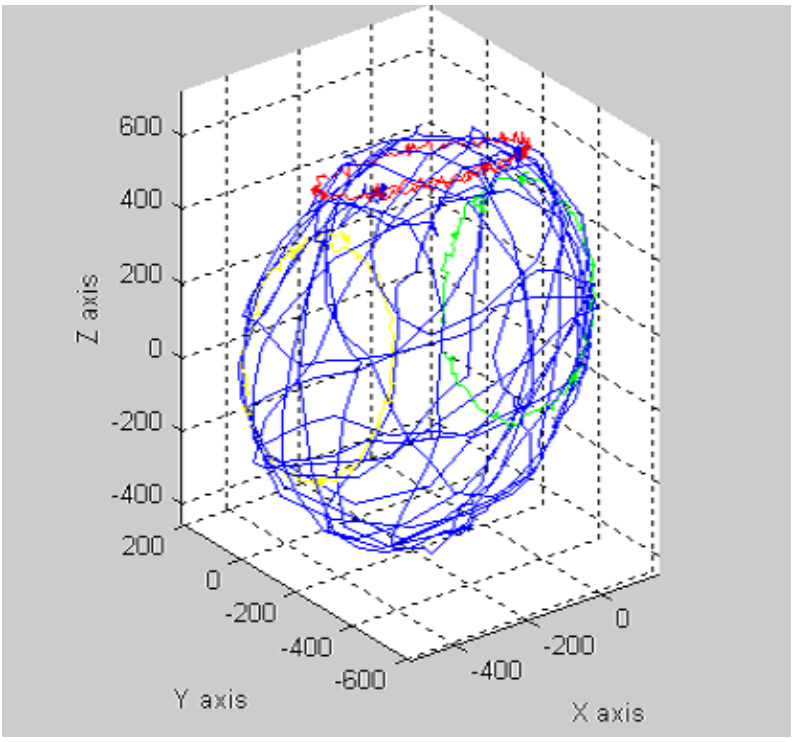
Obrázek 18 – Rychlost snímání [4]

Pro získání výsledného úhlu natočení je využita funkce *ArcTan()*. Je zde využit poměr naměřených hodnot v ose X a Y.



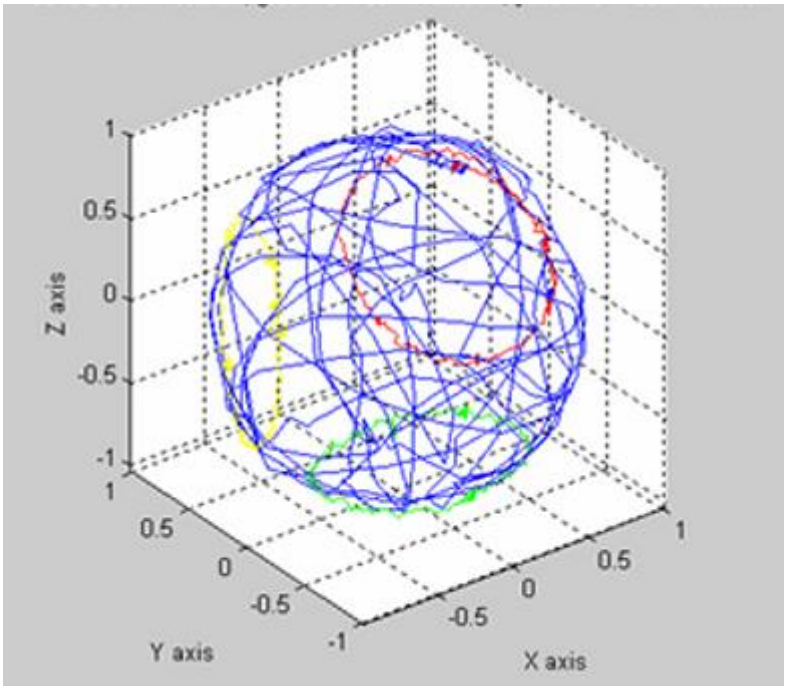
Obrázek 19 – Poměr os Y/X [4]

Pro získání relativně přesných dat je nejprve třeba kompas kalibrovat. Pokud vezmeme čistá data přímo z kompasu, v grafu vidíme ofsety způsobené okolními kovy, které mají neblahý vliv na přesnost měření. Magnetické pole je ovlivňováno předměty a zdroji magnetického pole v jeho těsném okolí (DPS, ostatní součástky, protékající proudy atd.). Kdy měkké okolí mění intenzitu zesilováním nebo zeslabováním magnetického pole, magneticky tvrdé okolí posouvá měření přičítáním vlastního magnetického pole.[21]



Obrázek 20 – Nepřesnost způsobena okolním prostředím [6]

Po kalibraci vypadají výstupní data zcela jinak:



Obrázek 21 – Nepřesnost způsobena okolním prostředím [6]

Pro kalibraci kompasu musíme nejdříve posbírat co nejvíce dat ze všech os. Tedy například při kalibraci osy X budeme chytat data tak dlouho, dokud neotočíme kompasem

tak,abychom nachytali data v rozmezí 360 stupňů (to platí pro každou osu zvlášť). Jakmile máme tyto hodnoty, hledáme v nich maximum a minimum. Tyto maxima a minima porovnáme s každou osou a pokud se neshodují, je potřeba kompas zkalibrovat, neboť nám okolní prostředí zaneslo do měření nějaké nežádoucí složky. Ofset musí být ve všech osách symetrický (tedy všechny osy mohou měřit špatně, ale stejně), jelikož vystupují v poměrech. Samotná kalibrace spočívá ve spuštění měření a otáčení desky všemi směry o plných 360° tak, aby došlo k plnému otočení ve všech možných směrech. [5]

Pro kalibraci kompasu jsem použil vlastní funkci:

```
void Calibrate_Comp(short * X_Offset, short * Y_Offset,double *Nasobek)
```

kde `X_Offset` a `Y_Offset` jsou ofsety způsobené magneticky tvrdými materiály a `Nasobek` je pro určení zesílení či zeslabení magnetického pole, což je způsobeno magneticky měkkými materiály.

Ofsety i hodnota pro zesílení po kalibraci ovlivňují hodnoty čtené z kompasu:

```
COMPAS.X=(short)((COMPAS.X_H<<8) | COMPAS.X_L)-X_Offset;//+X_Offset;
COMPAS.Y=(short)((COMPAS.Y_H<<8) | COMPAS.Y_L)-Y_Offset;//+Y_Offset;
```

Jelikož je důležité, aby poměry obou os byly stejné, stačí měnit zesílení pouze v jedné ose:

```
COMPAS.Y*=Nasobek;
```

Po správné kalibraci by ve všech osách a úhlech natočení kompasu měla být naměřena stejná maxima a minima.

Zjištění minima a maxima v osách X a Y:

```
if (MAX_Y<COMPAS.Y)MAX_Y=COMPAS.Y;
if (MAX_X<COMPAS.X)MAX_X=COMPAS.X;

if (MIN_X>COMPAS.X)MIN_X=COMPAS.X;
if (MIN_Y>COMPAS.Y)MIN_Y=COMPAS.Y;

 *X_Offset=MIN_X+MAX_X;     //posunuti
 *Y_Offset=MIN_Y+MAX_Y;

 *X_Offset-=(*X_Offset)/2;
 *Y_Offset-=(*Y_Offset)/2;
```

Zesílení:

```
*Nasobek=(double)(MAX_X-MIN_X)/(MAX_Y-MIN_Y);
```

Získání úhlu ve stupních se provádí následovně:

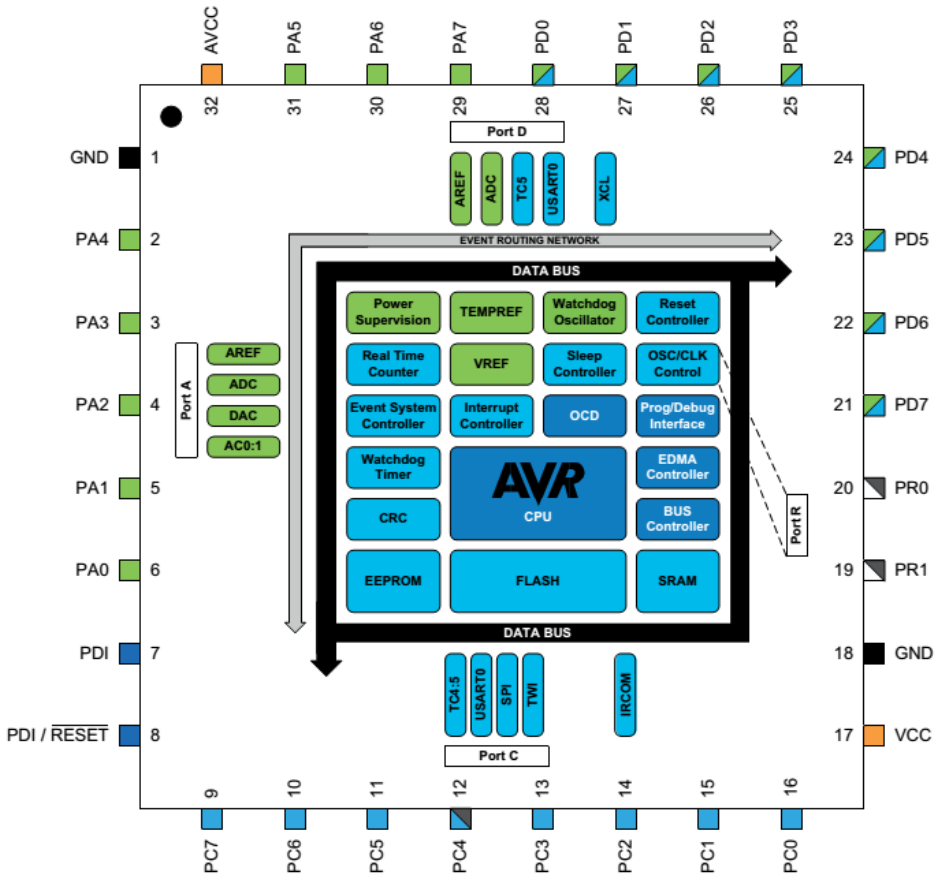
```
if (COMPAS.X==0)
{
    if (COMPAS.Y>0)
    {
          COMPAS.fHeading=90;
    }else
    {
          COMPAS.fHeading=270;
    }
}
else
{
    COMPAS.Y*=Nasobek;

    COMPAS.fHeading=(float)(atan2f(COMPAS.Y,COMPAS.X)*(180/3.14159265359));

}

if (COMPAS.fHeading<0)
{
    COMPAS.fHeading+=360;
}
```

2.5 Řídící mikrokontrolér Slave zařízení

S ohledem na budoucnost zařízení byl použit relativně nový (2013) mikrokontrolér od firmy Atmel ATxmega32E5 založený na AVR rozšířené RISC architektuře. Hlavní výhoda proti klasickým starším ATmega procesorům je jeho nynější celkově větší podpora s jistější budoucností. Xmega řady E je první série s 32 piny dostupné v nejmenší velikosti pouzdra 4 x 4 mm (QFN).

Navíc jsou ze všech Xmega řad nejúspornějšími mikrokontroléry a hodí se tak pro aplikace, kde je potřeba brát ohled na úsporu energie, tedy přístroje napájené baterií (akummulátorem).

AVR CPU kombinuje bohatý instrukční set (142 instrukcí) s 32 univerzálními pracovními registry, přímo napojených na aritmeticko logickou jednotku ALU umožňující nezávislý přístup ke dvěma registrům najednou v jedné instrukci vykonané v jednom hodinovém cyklu.



Obrázek 22 – Piny - AVR Xmega32E5 – [7]

MCU má vnitřní oscilátor 8MHz a energeticky méně náročný 32 KHz krystal pro Real Time aplikace. Maximální takt je možné nastavit až na 32 MHz , s kterým lze dosáhnout až 32 MIPS.

Doporučené napájecí napětí je 3.3 V, avšak rozsah je stanoven od 1.6 V do 3.6 V. Mikrokontrolér poskytuje širokou výbavu. Patří sem v systému přeprogramovatelná FLASH paměť o velikosti 32 KB (plus 4KB bootovací sekci), 1KB EEPROM pro zálohu dat i po odpojení napájení, 4 KB SRAM, osmiudálostní systém a programovatelné multi-úrovňové přerušení, dále 26 obecně vstupně výstupních pinů, jeden 16 – bitový časovač reálného času, další tři flexibilní 16 – bitové čítače/časovače, dvě USART sběrnice s možností funkcí Master SPI, jednu TWI (I2C) a jednu SPI sběrnici. Také obsahuje 12 – bitový A/D převodník s šesnácti kanály s rychlostí až 300 KS/s.

Klíčové parametry

• Flash (KBytes)	32
• Počet pinů	32
• Maximální výpočetní frekvence [MHz]	32
• CPU	8 – bit AVR
• Počet Touch kanálů	56
• Vstupně výstupní piny	26
• Externí přerušení	26

2.5.1 Vstupně výstupní piny

Xmega32E5 disponuje 26 I/O piny. Mohou být různě nastveny jako analogové či digitální vstupy /výstupy. Je zde možnost nastevení push-pull nebo open-colector. Nechybí zde samozřejmě možnost připojení vnitřního pull-up rezistoru. Na dvouch pinech lze také získat jako výstup analogovou hodnotu z 12-bitového D/A převodníku.

2.5.2 Čítače / Časovače

Použitý mikrokontrolér je vybavem celkem čtyřmi 16bitovými časovači s možností PWM nebo s komparačním módem, každý časovač je ještě rozdělěn na kanály. Kaskádním řazením lze dosáhnout jednoho 32- bitového časovače / čítače. Lze jej použít jako generátor frekvence, pulzně šířkové modulace PWM nebo také jako vstupní operace pro zachycení nějaké události, například pro spuštění A/D převodu.

2.5.3 USART

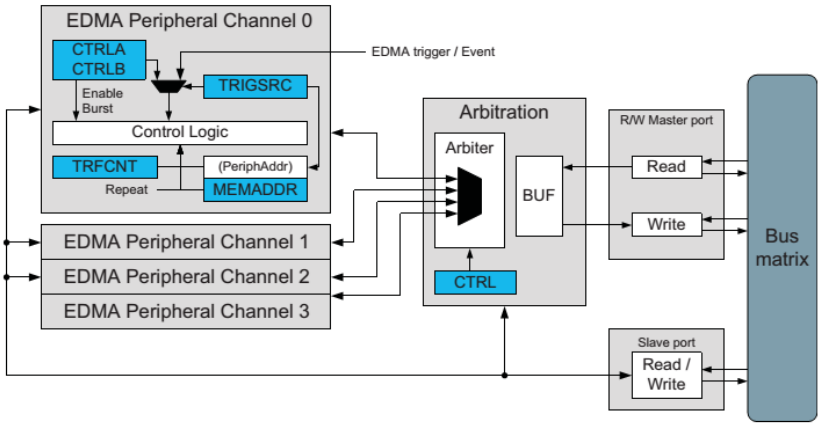
Plně-duplexní dvou drátový nebo polo-duplexní přenos může být použit na dvou kanálech zároveň a to zejména za použití EDMA řadiče. Synchronní přenos je podporován až do poloviny hodinové frekvence procesoru, asynchronní do *1/8* frekvence CPU. Podporuje 5,6,7,8 nebo 9 bitů dlouhý rámec pro přenos zakončený sudou nebo lichou paritou. Pro příjem nebo vyslání telegramu je možno použít přerušení ISR. Značnou výhodou je použití USARTU jako Master SPI sběrnici, kterou jsem využil na místo klasické SPI z důvodu rozmístění pinů.

2.5.4 EDMA

EDMA je funkce umožňující přenášet data s minimálním zásahem CPU.

- Z datové paměti do datové paměti
- Z datové paměti do periferie
- Z periferie do datové paměti
- Z periferie do periferie

Obsahuje 4 kanály s odděleným přenosem, přerušením a adresováním. Je možné přenášet celé rámce (bloky) dat od 1 B až po 64 KB s možností opakování tedy až do 128 KB.



Obrázek 23 – EDMA - AVR Xmega32E5 – [7]

V práci byla funkce EDMA využita pro příjem dat z GPS, neboť to velmi ušetří čas procesoru, který by jinak jako Slave zařízení stále musel obsluhovat sběrnici USART.

2.6 Řídící mikrokontrolér Master zařízení

Jelikož je na Master zařízení použit barevný grafický displej s rozlišením 320 x 240 bodů, bylo potřeba zvolit mikrokontrolér, který nebude mít problémy s rychlým vykreslováním údajů na displeji. Od firmy Atmel byl tedy vybrán mikrokontrolér SAM4S ARM Cortex-M4. Je založen na high-performance 32-bit ARM® Cortex®-M4 RISC procesoru s maximální hodinovou frekvencí 120 MHz. Disponuje 2048 KB Flash pamětí. Mezi periferie mimo jiné patří plně rychlostní USB 2.0 (12 Mbps) s vestavěným transcieverem. Také je připraven pro dotykové Touch vstupy.

Klíčové parametry:

- 2048 KB Flash paměť
- 16 KB ROM s vestavěným boot-loader
- DSP instrukční set
- Vestavěný napěťový detektor
- Quartz nebo keramický oscilátor 3 - 20 MHz, či nízkoodběrový 32.768 KHz pro RTC
- 2 PLL oscilátory (240 MHz) pro USB
- Teplotní senzor
- 22 DMA kanálů

2.6.1 Periferie

Na Master zařízení byly využity stejné periferie jako na Slave, navíc byla použita pouze TWI sběrnice pro vyčítání dat z elektronického kompasu.

2.6.1.1 USART

Sběrnice Usart byla využita pro komunikaci s GPS modulem. Rychlost přenosu je 9600 bps s jedním *Stop bitem* a žádnou paritou. Aby nebyl mikrokontrolér příliš zaměstnáván při příjmu jednotlivých bajtů od GPS, byla také využita funkce DMA.

Offset	Register	Name	Access	Reset
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read/Write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	0x0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	Receive Holding Register	US_RHR	Read-only	0x0
0x001C	Transmit Holding Register	US_THR	Write-only	–
0x0020	Baud Rate Generator Register	US_BRGR	Read/Write	0x0
0x0024	Receiver Time-out Register	US_RTOR	Read/Write	0x0
0x0028	Transmitter Timeguard Register	US_TTGR	Read/Write	0x0
0x2C–0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read/Write	0x0
0x0050	Manchester Configuration Register	US_MAN	Read/Write	0xB0011004
0x0054–0x005C	Reserved	–	–	–
0x0060–0x00E0	Reserved	–	–	–
0x00E4	Write Protection Mode Register	US_WPMR	Read/Write	0x0
0x00E8	Write Protection Status Register	US_WPSR	Read-only	0x0
0x00EC–0x00FC	Reserved	–	–	–
0x100–0x128	Reserved for PDC Registers	–	–	–

Obrázek 24 – Registry USART [10]

2.6.1.2 DMA

PDC (*Peripheral DMA Comtroller*) je implementována na USART tím způsobem, že ISR přerušení v programu nastane (se obslouží) až poté, co je přijmut celý telegram – v tomto případě jedna z NMEA vět. Tím je zaručeno menší zaměstnání výkonu procesoru, který v danou chvíli může být potřeba jinde.

Offset	Register	Name	Access	Reset
0x00	Receive Pointer Register	PERIPH[(1)]_RPR	Read/Write	0
0x04	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x08	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x0C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x10	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x14	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x18	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x1C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x20	Transfer Control Register	PERIPH_PTCR	Write-only	0
0x24	Transfer Status Register	PERIPH_PTSR	Read-only	0

Obrázek 25 – Registry DMA [10]

2.6.1.3 SPI

Pomocí SPI sběrnice je obsluhován barevný grafický displej a RF modul SX1276. Protože každé zařízení komunikuje na jiné přenosové rychlosti, bylo potřeba využít více SPI kanálů, kterými mikrokontrolér disponuje (s externím dekodérem až 15). Maximální přenosová rychost PSI je 60 Mbit/s při taktu procesoru 120 MHz. Tuto rychlost jsem využil pro vykreslování dat na TFT displeji. Pro komunikaci s SX1276 je použita přenosová rychost 5 Mbit/s. Vývody MISO MOSI a SCK jsou společné pro všechna použití SPI, proto je potřeba dané zařízení vybírat pomocí pinu *SELECT*.

2.6.1.4 TWI

Použitý elektronický kompas komunikuje pouze na TWI sběrnici a proto byla také využita. Výhodou sběrnice TWI je především využití pouze dvou vodičů, ovšem nevýhodou je poloduplexní komunikace a relativně malá přenosová rychlost - v tomto případě 400 Kbit/s.

2.7 Operační systém

V Master zařízení je potřeba obsluhovat mnoho událostí najednou a při klasickém rozvrhování celého programu by bylo velmi složité všechna kritická místa ošetřit vůči kolizím – i zkušený programátor by měl problémy vše uhlídat. Aby se celkové programování usnadnilo, bylo zde výhodné použít operační systém, v mém případě FreeRTOS™. RTOS je zkratka pro *Real Time Operating System.*

Použití OS usnadňuje práci právě pro možnost vytvoření několika úloh (*Task*), které běží nezávisle na sobě a pro uživatele se jeví tak, že všechny běží paralelně v jednom okamžiku. Kernel si samozřejmě v závislosti na jednotlivých prioritách řídí chod programu tak, že vždy běží právě jenom jedno vlákno. V mém programu jsem vytvořil 5 úloh a každá má stejnou prioritu.

2.7.1 Druhy OS

Existuje mnoho operačních systémů, které by se daly pro moji aplikaci využít. Každý má své specifické vlastnosti, ovšem pro základní použití se velmi podobají a pro mě je přednější podpora pro nasazení systému na daný mikrokontrolér než určité výhody každého z nich.

Nějčastěji používané operační systémy pro mikrokontroléry:

- FreeRTOS
- ChibiOS/RT
- RTX
- eCos

2.7.2 Real-time aplikace

Hlavní rozdíl mezi standardní a *Real-time* aplikací je časové omezení týkající se akcí, které mají provádět. V *Real-time* aplikaci čas, v který bude aplikace něco vykonávat, může být předurčen deterministicky na základě znalostí o použitém hardware a software.

Rozdělujeme na:

Soft real-time: aplikace, které mají daný nějaký časový termín do vykonání operace, ale v případě nedodržení určitého časového limitu se nestane nic vážného. Příkladem může být obsluha tlačítek – pokud se nestihne jeho obsluha vykonat včas, bude se nanejvýš zařízení chovat protivně.

Hard real-time: Je přesný opak Soft real-time, Pokud se nedodrží stanovený čas, může to mít fatální následky. Typicky se může jednat o spouštění airbagu při autonehodě, pokud systém nezareaguje ihned, je takřka nepoužitlený.

2.7.3 Multitasking

Nejzákladnější funkce společná pro všechny operační systémy je multitasking. K tomu může být přidána podpora pro práci se sítí, periférie, uživatelský interface, tisknutí atd.

Ne vždy jsou ale potřeba všechny tyto funkce, ale jen některé z nich. Většina systémů ve vývojových aplikací používá jenom několik základních funkcí z podpory pro multitasking. Velikost OS může kolísat ve velikosti použité paměti od 300 b až do 10Kb. Josu tedy dosatečně malé na to, aby se daly nasadit do interní paměti mikrokontroléru. [7]

Vestavěné systémy obvykle mají přístup pouze k jednomu procesoru, který obsluhuje spousty vstupů a výstupů. Real Time OS musí rozdělit čas mezi proměnné aktivity tak, že všechny budou vyřízeny v závislosti na jejich nastavených prioritách.

Real Time Operating vždy obsahují:

- Podporu pro více úloh běžících současně
- Plánovač k určení, která úloha má běžet
- Schopnost komunikovat mezi úlohami

2.7.4 FreeRTOS

Jelikož jsem psal program v Atmel Studiu, využil jsem operačního systému FreeRTOS, který společnost Atmel připravila přímo (nejen) pro můj mikrokontrolér SAM4S ARM Cortex-M4 a jeho implementace je s využitím ASF (*Atmel Software Framework*) velmi jednoduchá. [7]

Podpora pro Cortex-M4 zahrnuje všechny standardní funkce FreeRTOS:

- Preemptivní nebo kooperativní operace
- Přizpůsobení priorit úloh
- Fronty
- Binární semafory
- Počítací semafory
- Rekurzivní semafory

- Mutexy
- *Tick Hook* funkce
- *Idle Hook* funkce
- Kontrola přetečení zásobníku

2.7.4.1 Management úloh

FreeRTOS umožňuje nelimitovaný počet běžících úloh v závislosti na velikosti mu poskytnuté paměti.

Každý Task, který je založen, má alokovanou specifickou část v paměti RAM, která se nazývá *HEAP*. Velikost této sekce se nastavuje při vytváření Tasku. Heap obsahuje soubory pro Task a TCB (Task Control Board), který umožnuje, aby mohl být ovládán systémem.

FreeRTOS umí zacházet s více úlohami souběžně, ale právě pouze jedna aplikace může běžet v jednom okamžiku na jednojádrovém procesoru. Systém, který úlohy přepíná, se jmenuje plánovač – *Scheduler*. Scheduler je část jádra FreeRTOS, která se stará o přepínání mezi úlohami podle dané priority. [7]

2.7.5 Vytovořené Tasky

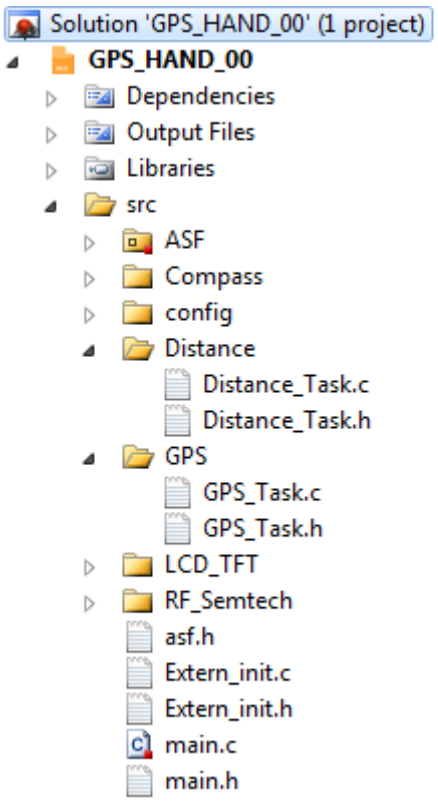
Prvně je potřeba jednotlivé úlohy vytvořit, praktická ukázka viz níže:

```
/*Create GPS Task*/
xTaskCreate(GPS_Task,"GPS",configMINIMAL_STACK_SIZE+800,NULL, 1,&GPS_id);
/*Create Semtech Task*/
xTaskCreate(RF_Task,"sx1276",configMINIMAL_STACK_SIZE+800,NULL, 1,&Semtech_id);
/*Create Compass Task*/
xTaskCreate(Compass_Task,"LIS3DH",configMINIMAL_STACK_SIZE+200,NULL, 1,&Compass_id);
/*Create LCD Task*/
xTaskCreate(LCD_Task,"LCD",configMINIMAL_STACK_SIZE+800,NULL, 1,&Lcd_id);
/*Create Distance calc Task*/
xTaskCreate(Distance_Task,"Distance",configMINIMAL_STACK_SIZE+200,NULL,1,&Distance_i
d);
```

Zde můžeme vidět pět vytvořených tásků. Pro utvoření je potřeba zadat název (`GPS_Task`), jméno pro lepší identifikaci v kódu (`"GPS"`), velikost alokované v paměti – Heap (`configMINIMAL_STACK_SIZE+800`), vstupní parametry – v tomto případě nepoužito (`NULL`), přidělení priority (1) a task id (`,&GPS_id`) například pro použití ve funkci `vTaskResume(Gps_id);`

2.8 Ukázka kódu – programu

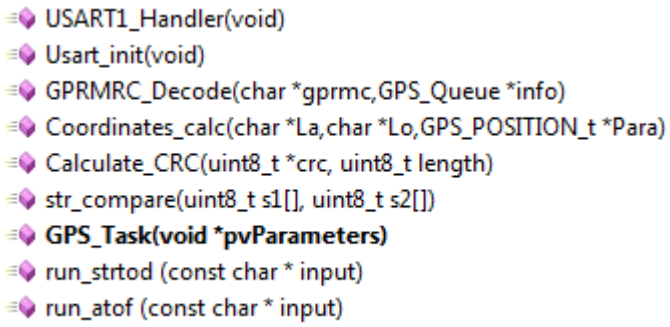
Obsáhlost kódu mě donutila pro lepší přehlednost rozdělit jednotlivé úlohy do vlastních složek a dále do .C a .H souborů.



Obrázek 26 – Rozvržení souborů v programu

2.8.1 GPS_Task

Při prvním vstupu do tasku jsou deklarovány všechny proměnné, které budeme chtít používat. Jediným způsobem pro předávání dat mezi tasky, je využití front - neexistuje totiž použití nějaké globální proměnné, která bude vidět ve všech úlohách. Na řádku čtyři je vidět deklarace fronty pro LCD, díky této frontě bude moci daná úloha GPS_Task schopna komunikovat s úlohou LCD_Task a předávat jí tak data k zobrazení na displeji. Řádek 5 deklaruje frontu pro příjem dat z ISR USART přerušení a řádek 6 frontu pro odesílná souřadnic do úlohy Distance_Task. Dále následuje inicializace sběrnice USART funkcí Usart_init();



Obrázek 27 – Funkce použité v souboru GPS_Task

```
1.  /*********************************************/
2.  void GPS_Task(void *pvParameters)
3.  {
4.      LCD_Queue LCD;
5.      GPS_Queue DataToGPS;
6.      GPS_POSITION_t sGPS_MasterPosition;
7.
8.      vSemaphoreCreateBinary(GPS_Busy);
9.
10.     Usart_init();
11. //  Gps_init();
12.
13.     for (;;)
14.     {
15.
16.
17.         if((xSemaphoreTake(GPS_Busy,2000)==pdPASS))
18.         {
19.
20.             if(xQueueReceive(Queue_GPS,&DataToGPS,6000)==pdPASS)
21.             {
22.                 //pdca_channel_disable(PDCA_RX_CHANNEL);
23.                 //Pokud Byla nalezena veta
24.                 if (DataToGPS.GL_flag_gps==SENTENCE_OK)
25.                 {
26.                     if ((str_compare(GPS_GPRMC,DataToGPS.Nmea_data)==0))//
27.                     {
28.                         DataToGPS.Nmea_crc[0]=DataToGPS.Nmea_data[strlen(DataToGPS.Nmea_dat
    a)-MSB_CRC_CONST];
29.                         DataToGPS.Nmea_crc[1]=DataToGPS.Nmea_data[strlen(DataToGPS.Nmea_dat
    a)-LSB_CRC_CONST];
30.
31.                         DataToGPS.My_crc[2]=Calculate_CRC(&DataToGPS.Nmea_data[0],(strlen(D
    ataToGPS.Nmea_data)-UNTIL_CRC_CONST));   // CRC
32.
33.                         DataToGPS.My_crc[0]=0xf0 & DataToGPS.My_crc[2];     //mSB cislo pre
    vedu na string
34.                         DataToGPS.My_crc[1]=0x0f & DataToGPS.My_crc[2];     //lSB cislo pre
    vedu na string
35.
36.                         my_itoa_3((int)DataToGPS.My_crc[0],&DataToGPS.My_str_crc[0],0xF);
37.                         my_itoa_3((int)DataToGPS.My_crc[1],&DataToGPS.My_str_crc[1],0xF);
38.
39.                         if (DataToGPS.My_str_crc[0]>='a') DataToGPS.My_str_crc[0]-=('a'-
    'A');
40.                         if (DataToGPS.My_str_crc[1]>='a') DataToGPS.My_str_crc[1]-=('a'-
    'A');
41.
```

```
                        if ((DataToGPS.Nmea_crc[0]==DataToGPS.My_str_crc[0])&&(DataToGPS.Nm
    ea_crc[1]==DataToGPS.My_str_crc[1]))
                        {
                            GPRMRC_Decode(&DataToGPS.Nmea_data,&DataToGPS);
                            sGPS_MasterPosition.dLatitude=DataToGPS.dLatitude;
                            sGPS_MasterPosition.dLongitude=DataToGPS.dLongtitide;
                            xQueueSend(Queue_DIST_Master,&sGPS_MasterPosition,100);

                            LCD.Task_id=GPS_i;
                            LCD.Text=DataToGPS.Time;
                            LCD.Text2="Hodiny";
                            //LCD.S_number[0]=(short)DataToGPS.Speed_km;
                            xQueueSend(Queue_LCD,&LCD,1000);


                        }else //nesedi CRC
                        {
                            LCD.Task_id=GPS_i;
                            LCD.Text2="CRC Fail";
                            LCD.S_number[0]=(short)DataToGPS.Speed_km;
                            xQueueSend(Queue_LCD,&LCD,0);
                        }

                    }else if (str_compare(GPS_GPTXT,DataToGPS.Nmea_data)==0)
                    {
                        LCD.Task_id=GPS_i;
                        LCD.Text="GPTXT";
                        xQueueSend(Queue_LCD,&LCD,100);
                    }


                }

                 usart_enable_interrupt(CONF_UART,US_IER_RXRDY);

            }else // data z fronty nejsou
            {
                LCD.Task_id=GPS_i;
                LCD.Text="fronta GPS prazdna";
                //LCD.S_number[0]=(short)DataToGPS.Speed_km;
                xQueueSend(Queue_LCD,&LCD,100);
                vTaskDelay(1000/portTICK_RATE_MS);
                 usart_enable_interrupt(CONF_UART,US_IER_RXRDY);

            }


        }else
        {
            LCD.Task_id=GPS_i;
            LCD.Text="GPS - nepripojena";
            //LCD.S_number[0]=(short)DataToGPS.Speed_km;
            xQueueSend(Queue_LCD,&LCD,100);
            vTaskDelay(1000/portTICK_RATE_MS);
             usart_enable_interrupt(CONF_UART,US_IER_RXRDY);

        }


    }

}
```

Řádek 8 dává vzniknout binárnímu semaforu , pro pozastavení GPS_Task. Následuje nekonečná smyčka for (řádek 13). Po vstupu hned čekáme na "povolení " pro rozběhnutí úlohy, jelikož GPS_Task má za úkol zpracovat přijatá data z GPS pomocí USARTU, čeká se až v ISR přerušení je odchycena celá věta (např. GPRMC), poté se příkazem `xSemaphoreGiveFromISR(GPS_Busy,pdTRUE)` "odevzdá" semafor a GPS_Task může pokračovat. Parametr `pdTRUE` ve funkci znamená, že po odevzdání se ještě nejdříve dokončí celá ISR funkce a až poté bude moci zase GPS_Task pokračovat. Pokud by zde byl `pdFALSE` – ISR routine se nedokončí a hned by GPS_Task mohl pracovat.

Řádek 13 (`if((xSemaphoreTake(GPS_Busy,2000)==pdPASS))`) má jako vstupní parametr číslo 2000 – tzn. že pokud po 2000 ticích úloha neobdrží povolení k pokračování (v tomto případě se jedná o příjem dat z GPS), například nebude připojena GPS, nebo z nějakého důvodu MCU z ní neobdrží data, program skočí na řádek 89. Zde do fronty pro LCD zapíšeme informaci, že GPS není připojena, frontu odešleme s platností 100 ticků (například pokud bude fronta pro LCD plná, bude se čekat pouze vymezený čas a jestliže se do té doby fronta neuvolní, tak se data zahodí). Dále na řádku 95 `vTaskDelay(1000/portTICK_RATE_MS)` zajistíme, aby se Kernel vrátil do této úlohy až za 1000 ms – zajistíme tím tak plynulost běhu programu. Jelikož GPS generuje data v intervalu jedné sekundy, bylo by zbytečné, aby se Kernel stále dotazoval, zda už byla nová data z GPS přijata dříve, než právě za jednu sekundu.

Řádek 20 se dotazuje, zda jsou ve frontě Queue_GPS nějaká data, pokud ano, program pokračuje na řádek 24. Můžeme znovu vidět `if(xQueueReceive(Queue_GPS,&DataToGPS,6000)==pdPASS)` parametr 6000, který udává jak dlouhou dobu se má čekat na data, pokud data nepřichází po určitou dobu, program pokračuje na řádek 76, kde opět pošleme skrze frontu informace pro vypsání na LCD.

Řádek 26 se již stará o samotné zpracování přijatých dat (NMEA vět). Patří sem především rozklíčování jednotlivých údajů jako je například rychlost pohybu, hodiny, souřadnice či nadmořská výška – to zajišťuje funkce `GPRMRC_Decode(&DataToGPS.Nmea_data,&DataToGPS);` (řádek 44). Po vypreparování souřadnic z věty je hned pošlu do úlohy s názvem Distance_Task s okamžitou platností 100 ticků, zároveň nechám na TFT displeji zobrazit aktuální šas v UTC pásmu.

2.8.2 Funkce GPRMC_Decode:

```
1.  void GPRMRC_Decode(char *gprmc,GPS_Queue *info)
2.  {
3.      char La[12];
4.      char Lo[13];
5.      /*getting info->DataToGPS.Time*/
6.      for(uint8_t i=0;i<6;i++)
7.      {
8.          info->Time[i]=gprmc[i+7];
9.      }
10.     info->Time[8]=0;
11.     info->Time[7]=info->Time[5];
12.     info->Time[6]=info->Time[4];
13.     info->Time[5]=':';
14.     info->Time[4]=info->Time[3];
15.     info->Time[3]=info->Time[2];
16.     info->Time[2]=':';
17.
18.
19.     /*getting Latitude*/
20.     for(uint8_t i=0;i<11;i++)
21.     {
22.         La[i]=gprmc[i+20];
23.         Lo[i]=gprmc[i+32];
24.     }
25.
26.     // ddmm.mmmmm,dddmm.mmmm
27.     uint32_t dwDegrees = 0;
28.     dwDegrees = (La[0]-'0')*10*60*10000  + (La[1]-'0')*60*10000 +   (La[2]-
    '0')*100000        + (La[3]-'0')*10000 +
29.     (La[5]-'0')*1000          + (La[6]-'0')*100 +
30.     (La[7]-'0')*10            + (La[8]-'0');
31.
32.     info->dLatitude = ((double)dwDegrees/(60*10000));
33.
34.     dwDegrees = 0;
35.     dwDegrees = (Lo[0]-'0')*10*10*60*10000 + (Lo[1]-'0')*10*60*10000  + (Lo[2]-
    '0')*60*10000 +  (Lo[3]-'0')*100000 + (Lo[4]-'0')*10000 +
36.     (Lo[6]-'0')*1000   + (Lo[7]-'0')*100    +
37.     (Lo[8]-'0')*10     + (Lo[9]-'0');
38.
39.     info->dLongtitide = ((double)dwDegrees/(60*10000));
40.
41.     for(uint8_t i=0;i<4;i++)
42.     {
43.         if(i!=1)info->Speed_knots[i]=gprmc[i+45];
44.
45.     }
46.         info->Speed_knots[4]=0;
47.
48.
49.     //  info->DataToGPS.Latitude[9]='\n';
50.     //  info->DataToGPS.Longtitude[9]='\n';
51.
52. }
```

Z kódu je vidět pouze začlenění dat z jednoho velkého pole (NMEA věty) do různých malých polí – např. hodin do pole `info->Time`.

2.8.3 Funkce ISR – USART1_Handler

```
1.  void USART1_Handler(void)
2.  {
3.      static short counter=0;
4.      irqflags_t LocSREG;
5.      LocSREG=cpu_irq_save();
6.      uint32_t ul_status;
7.      static GPS_Queue DataToGPS;
8.
9.
10.     taskENTER_CRITICAL();
11.
12.     usart_getchar(CONF_UART,&DataToGPS.Nmea_data[counter]);
13.
14.     if (DataToGPS.Nmea_data[counter]=='\n')
15.     {
16.         usart_disable_interrupt(CONF_UART,US_IER_RXRDY);
17.         DataToGPS.Nmea_data[counter]=0; // pro praci se strlen -
18.         counter=0;
19.         DataToGPS.GL_flag_gps=SENTENCE_OK;
20.         taskEXIT_CRITICAL();
21.         cpu_irq_restore(LocSREG);
22.         xSemaphoreGiveFromISR(GPS_Busy,pdTRUE); //dokonci se IRQ
23.         xQueueSendFromISR(Queue_GPS,&DataToGPS,portMAX_DELAY);
24.
25.
26.     }else
27.     {
28.         if ((counter<77)&&(DataToGPS.Nmea_data[0]==0x24)) counter++;//
29.         taskEXIT_CRITICAL();
30.         cpu_irq_restore(LocSREG);
31.
32.     }
33.
34.
35.
36. }
```

Princip této funkce je velmi primitivní. Vždy, když přijde nějaký znak od USARTU, je přečten a uložen do pole DataToGPS.Nmea_data. Ukládá se tak dlouho, dokud není přijat znak '\n' – tedy zakončovací znak NMEA věty. Po obdržení tohoto znaku se zakáže na řádku 16 přerušení od USARTU a celá nachytaná věta se předá do fronty, která ji odešle do úlohy GPS_Task.

2.8.4 Funkce Coordinates_calc

Tato funkce převádí souřadnice přijaté z GPS do správného formátu:

```
1.  void Coordinates_calc(char *La,char *Lo,GPS_POSITION_t *Para)
2.  {
3.      // ddmm.mmmmm,dddmm.mmmm
4.      uint32_t dwDegrees = 0;
5.      dwDegrees = (La[0]-'0')*10*60*10000  + (La[1]-'0')*60*10000 +   (La[2]-
    '0')*100000        + (La[3]-'0')*10000 +
6.      (La[5]-'0')*1000          + (La[6]-'0')*100 +
7.      (La[7]-'0')*10            + (La[8]-'0');
8.
9.      Para->dLatitude = ((double)dwDegrees/(60*10000));
10.
11.     dwDegrees = 0;
12.     dwDegrees = (Lo[0]-'0')*10*10*60*10000 + (Lo[1]-'0')*10*60*10000  + (Lo[2]-
    '0')*60*10000 +  (Lo[3]-'0')*100000 + (Lo[4]-'0')*10000 +
13.     (Lo[6]-'0')*1000   + (Lo[7]-'0')*100    +
14.     (Lo[8]-'0')*10     + (Lo[9]-'0');
15.
16.     Para->dLongitude = ((double)dwDegrees/(60*10000));
17.
18. }
19. /***
```

2.9 Výpočet vzdálenosti mezi stanicemi

Hlavní funkce celého projektu je určení vzdálenosti mezi stanicem Master a Slave. Obě stanice posílají do Distance Task svoje souřadnice, z kterých se vzdálenost vypočítá.

Nejjednodušeji by šlo využít Pythagorovu větu – ovšem zde nastává zásadní problém v nepřesnosti, neboť v rovině můžeme říci, že součet vnitřních úhlů v trojúhleníku je 180 stupňů, což neplatí na zakřiveném povrchu. V našem případě by se nejednalo o kritickou chybu, neboť na vzdálenost kolem 5 km chyba nebude velká. Se vzrůstající vzdáleností mezi body by se však odchylka razantně zvětšovala (zakřivení Země).

Proto byl využit výpočet pro Elipsoid WGS – 84. Pseudo kódy pro výpočet jsou veřejně dostupné na internetu.

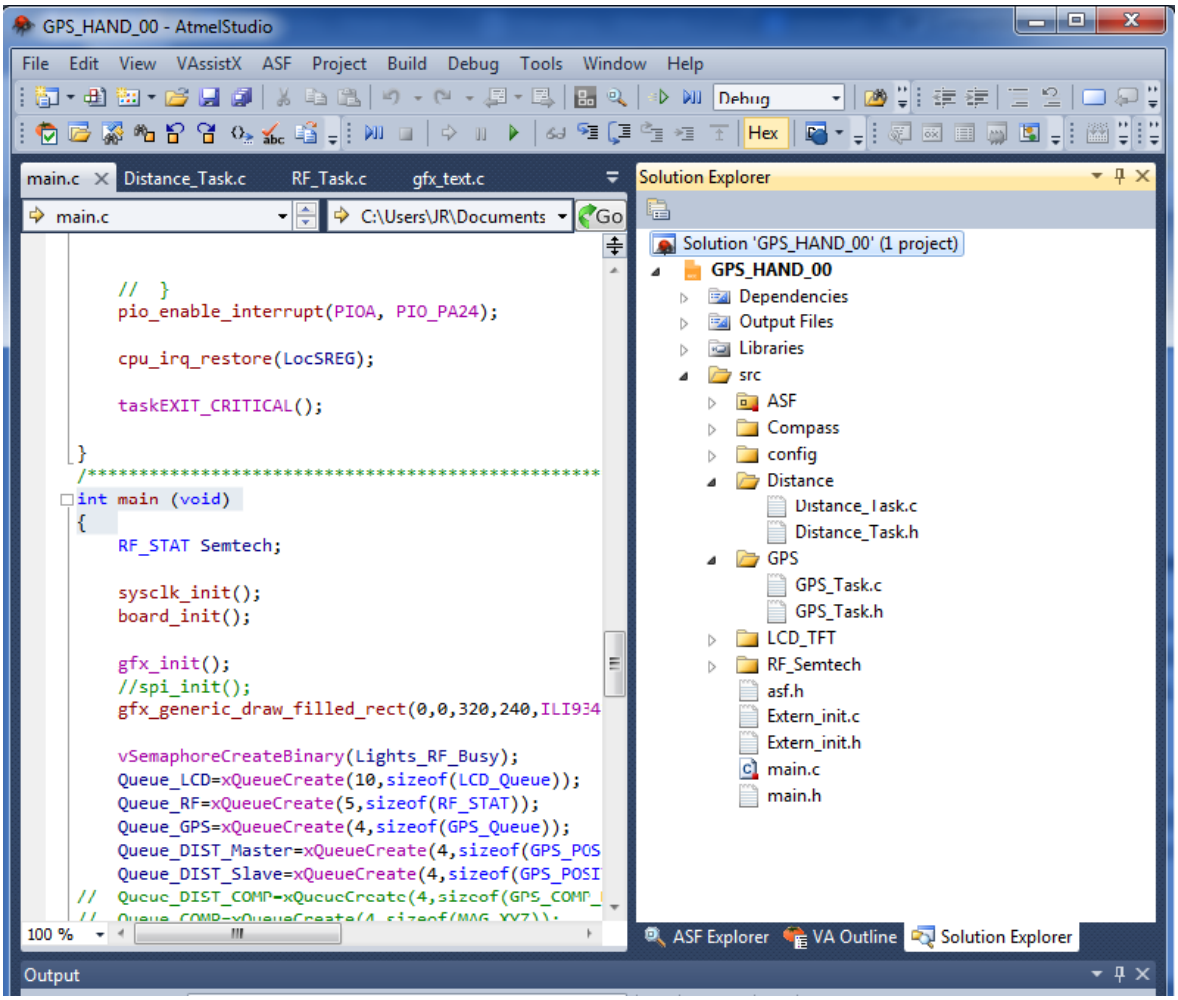
```
1.  char GPS_Utils_CalcDisAndBear( GPS_POSITION_t* psGPS_PositionMaster, GPS_POSITION
    _t* psGPS_PositionSlave, GPS_COMP_DATA_t* psCOMP_Data )
2.  {
3.      //------------------------------------------------------------------
4.      // WGS-84 ellipsoid params
5.      double a = 6378137;
6.      double b = 6356752.314245;
7.      double f = 1/298.257223563;
8.      //------------------------------------------------------------------
9.      double radtodeg = 57.29577951;
10.     double L = (psGPS_PositionSlave->dLongitude - psGPS_PositionMaster-
    >dLongitude)/radtodeg;  // degrees to rad
11.     double U1 = atan( (1-f) * tan( ( psGPS_PositionMaster-
    >dLatitude/radtodeg) ) );
12.     double U2 = atan( (1-f) * tan( ( psGPS_PositionSlave-
    >dLatitude /radtodeg) ) );
```

```
13.     double sinU1 = sin(U1);
14.     double sinU2 = sin(U2);
15.     double cosU1 = cos(U1);
16.     double cosU2 = cos(U2);
17.     //------------------------------------------------------------------------
18.     double lambda = L;
19.     double lambdaP = 0.0;
20.     uint32_t iterLimit = 100;
21.     double cosSqAlpha = 0;
22.     double cos2SigmaM = 0;
23.     double cosSigma = 0;
24.     double sigma = 0;
25.     double sinSigma = 0;
26.     double sinLambda = 0;
27.     double cosLambda = 0;
28.
29.     do
30.     {
31.         sinLambda = sin(lambda);
32.         cosLambda = cos(lambda);
33.         sinSigma  = sqrt( (cosU2*sinLambda) * (cosU2*sinLambda) +
34.         (cosU1*sinU2-sinU1*cosU2*cosLambda) * (cosU1*sinU2-
    sinU1*cosU2*cosLambda));
35.         //-- UnSafe comparing two float values -
36.         if ( 0.0 == sinSigma )
37.         {
38.             psCOMP_Data->dDistance = 0.0;
39.             psCOMP_Data->dBear     = 0.0;
40.             return (GPS_OK);; // co-incident points
41.         }
42.
43.         cosSigma = sinU1*sinU2 + cosU1*cosU2*cosLambda;
44.         sigma = atan2(sinSigma, cosSigma);
45.         double sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
46.         cosSqAlpha = 1 - sinAlpha*sinAlpha;
47.         cos2SigmaM = cosSigma - 2*sinU1*sinU2/cosSqAlpha;
48.         if (isnan(cos2SigmaM)) cos2SigmaM = 0;  // equatorial line: cosSqAlpha=0
    (Â§6)
49.         double C = f/16*cosSqAlpha*(4+f*(4-3*cosSqAlpha));
50.         lambdaP = lambda;
51.         lambda = L + (1-C) * f * sinAlpha *
52.         (sigma + C*sinSigma*(cos2SigmaM+C*cosSigma*(-
    1+2*cos2SigmaM*cos2SigmaM)));
53.     } while ( fabs(lambda - lambdaP) > 1e-12 && --iterLimit > 0);
54.
55.     if ( 0 == iterLimit ) return 0 ;  // formula failed to converge
56.
57.     double uSq = cosSqAlpha * (a*a - b*b) / (b*b);
58.     double A = 1 + uSq/16384*(4096+uSq*(-768+uSq*(320-175*uSq)));
59.     double B = uSq/1024 * (256+uSq*(-128+uSq*(74-47*uSq)));
60.     double deltaSigma = B*sinSigma*(cos2SigmaM+B/4*(cosSigma*(-
    1+2*cos2SigmaM*cos2SigmaM)-
61.     B/6*cos2SigmaM*(-3+4*sinSigma*sinSigma)*(-3+4*cos2SigmaM*cos2SigmaM)));
62.
63.     psCOMP_Data->dDistance = b*A*(sigma-deltaSigma);
64.     psCOMP_Data->dBear = atan2( cosU2*sinLambda, (cosU1*sinU2-
    sinU1*cosU2*cosLambda)) * radtodeg;
65.     //psCOMP_Data->dBear = psCOMP_Data->dBear - atan2((psGPS_PositionSlave-
    >dLatitude - psGPS_PositionMaster->dLatitude),(psGPS_PositionSlave->dLongitude -
     psGPS_PositionMaster->dLongitude)) * radtodeg;
66.
67.
68.     return(GPS_OK);
69. }
```

3 Vývojové prostředí

Vzhledem k použitým mikrokontrolérům od firmy Atmel jsem použil zdarma dostupné vývojové prostředí Atmel Studio 6.2. Jedná se plnohodnotné prostředí pro psaní kódu jak v assembleru tak v jazyce C. Grafické prvky jsou použity z prostředí Microsoft Visual Studio 2010.

Mezi hlavní výhody patří dostupnost – freeware, pro uživatele velice příjemné zpracování, online dubugging a přímá kompatibilita s vývojovým kitem SAM4S Explained, který byl v práci použit.



Obrázek 28 – Atmel Studio 6.2

3.1 ASF – Atmel Software Framework

ASF je nástavba od Atmelu, která zjednodušuje a celkově zpřehledňuje veškerou práci s programováním. Při samotném psaní programu není nutné možnost ASF využít a lze tak psát program i bez této podpory. Ovšem při velkých projektech se člověk velmi snadno začne ztrácet i ve svém vlastním kódu a právě tato nástavba má tento problém utlumit.

3.1.1 ASF – Příklad použití

Pokud bychom chtěli na mikrokontroléru Xmega32e5 nastavit na určitém pinu jistou logickou úroveň, tak bez ASF bychom to provedli následovně:

```
PORTA.OUTSET=0b1;    //nastaví LOG1 na PORTA.0
```

S použitím ASF:

```
ioport_set_pin_level(PORTA_PIN0CTRL,1); //nastaví LOG1 na PORTA.0
```

kde je `PORTA_PIN0CTRL` definován jako :

```
#define PORTA_PIN0CTRL     IOPORT_CREATE_PIN(PORTA,0)
```

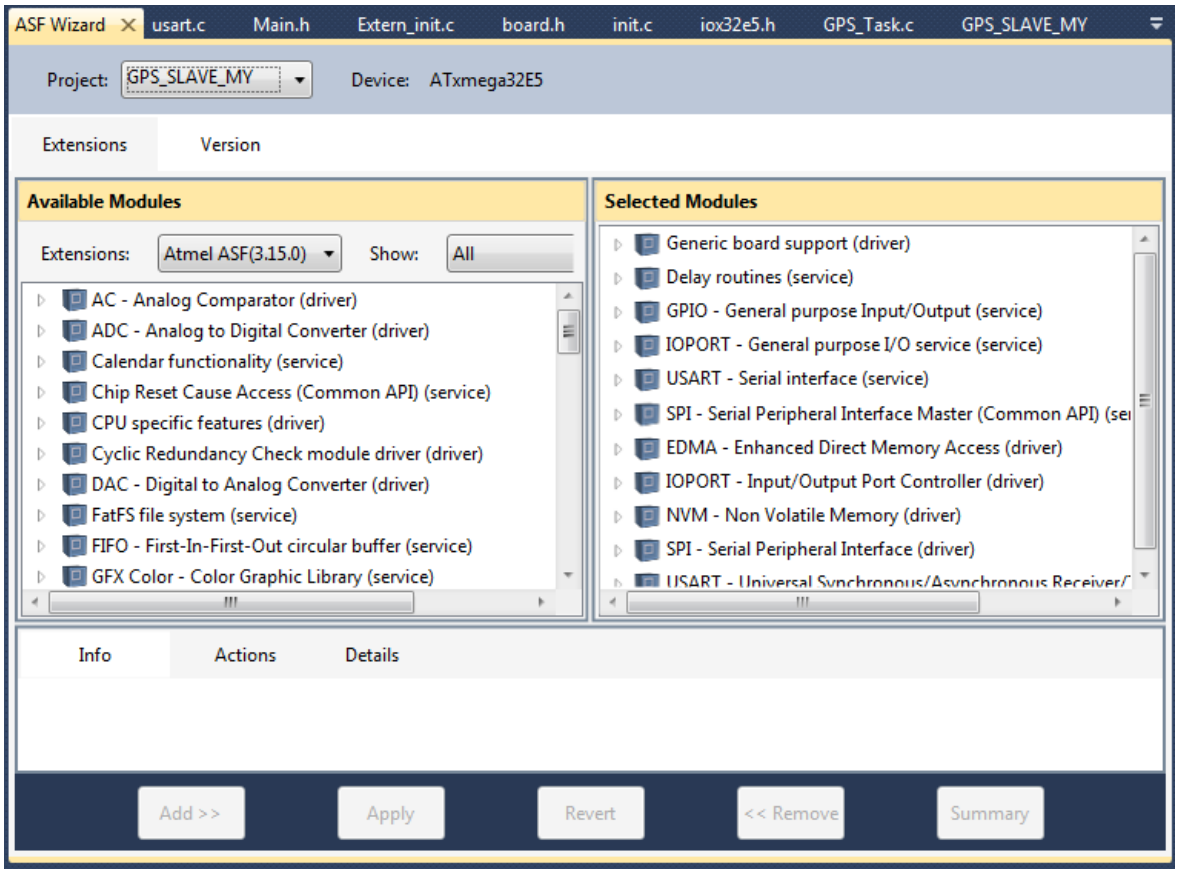
V tomto případě není ulehčení programu nějak výrazné, situace se ale velmi rychle změní, pokud budeme chtít nastavit například jednotku USART:

```
#define USART_SERIAL_EXAMPLE              &USARTD0
#define USART_SERIAL_EXAMPLE_BAUDRATE     9600
#define USART_SERIAL_CHAR_LENGTH          USART_CHSIZE_8BIT_gc
#define USART_SERIAL_PARITY               USART_PMODE_DISABLED_gc
#define USART_SERIAL_STOP_BIT             false

static usart_rs232_options_t USART_SERIAL_OPTIONS = {
                    .baudrate = USART_SERIAL_EXAMPLE_BAUDRATE,
                    .charlength = USART_SERIAL_CHAR_LENGTH,
                    .paritytype = USART_SERIAL_PARITY,
                    .stopbits = USART_SERIAL_STOP_BIT,
            };

/* Initialize usart driver in RS232 mode */
 usart_init_rs232(USART_SERIAL_EXAMPLE, &USART_SERIAL_OPTIONS);
```

Takto vypadá funkce `usart_init_rs232` uvnitř:

```
bool usart_init_rs232(USART_t *usart, const usart_rs232_options_t *opt)
{
       bool result;
       sysclk_enable_peripheral_clock(usart);
       usart_set_mode(usart, USART_CMODE_ASYNCHRONOUS_gc);
       usart_format_set(usart, opt->charlength, opt->paritytype,opt->stopbits);
       result = usart_set_baudrate(usart, opt->baudrate, sysclk_get_per_hz());
       usart_tx_enable(usart);
       usart_rx_enable(usart);

       return result;
}
```

Jedinečnou výhodou je opravdu velké urychlení programování, každý MCU má pro sebe ASF knihovny přizpůsobeny a ve většině případů tedy mohu vzít takto napsanou funkci pro Xmegu32E5 a při aplikování například na Xmega16D4 ji mohu pouze zkopírovat a hned bude jednotka USART fungovat.



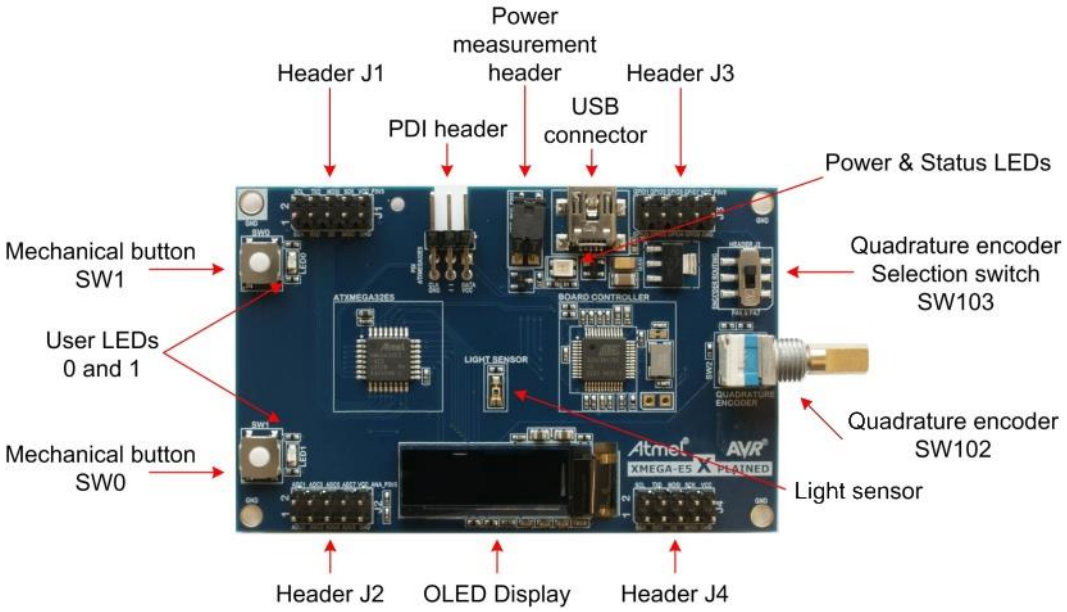
Obrázek 29 – ASF - Implementace

4 Praktická část

Jelikož se jedná o pouze beta verzi projektu a celkové použití jednotlivých prvků ještě není úplně rozhodnuto a tedy pro případnou změnu GPS modulu z L80 na L50 by se muselo předělávat schéma a tím i vyrobit novou desku DPS, nebylo zhotoveno žádné schéma, ale vše je zatím postaveno na kitech, které jsou spolu různě propojeny. Slave zařízení obsahuje pouze tři kity: GPS modul L80, Xmega32e5 Xplained Kit a RF Semtech 1276 kit.

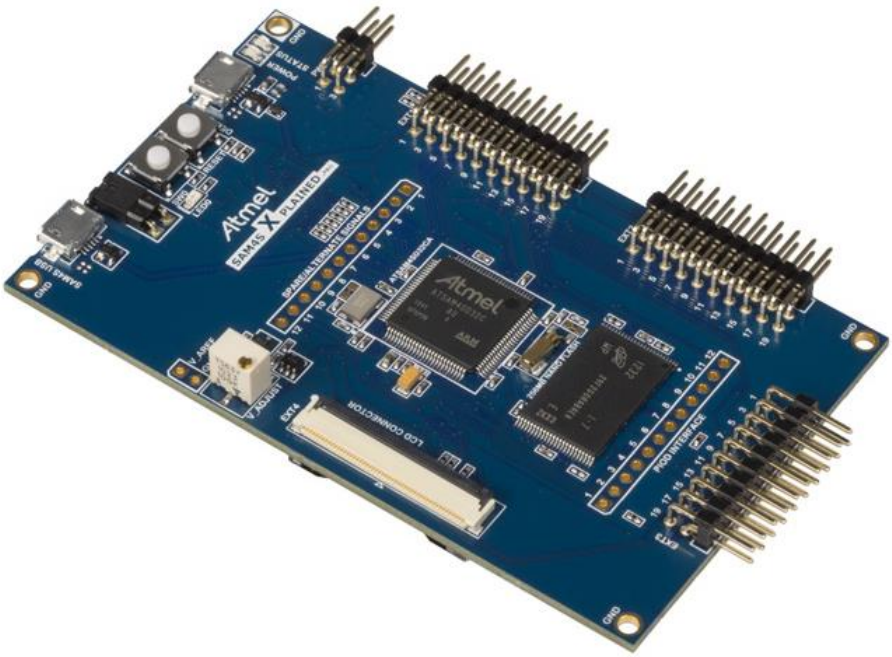
Na Master zařízení je použito kitů více: Atmel ARM SAM4S Xplained Pro, GPS modul L80, kompas – STEVAL-MK124V1, RF Semtech 1276 kit a TFT LCD barevný displej s rozlišením 320 x 240 bodů.

4.1 Slave Evaluation Kit XMEGA-E5 Xplained



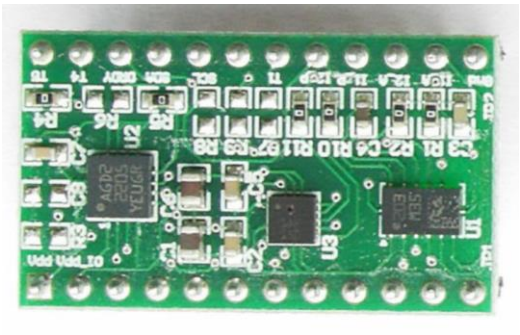
Obrázek 30 – Vývojový kit – Slave [9]

4.2 Master Xplained Pro Evaluation kit ATSAM4S-XPRO



Obrázek 31 – Vývojový kit – Master [10]

4.3 Kompas – STEVAL-MKI124V1

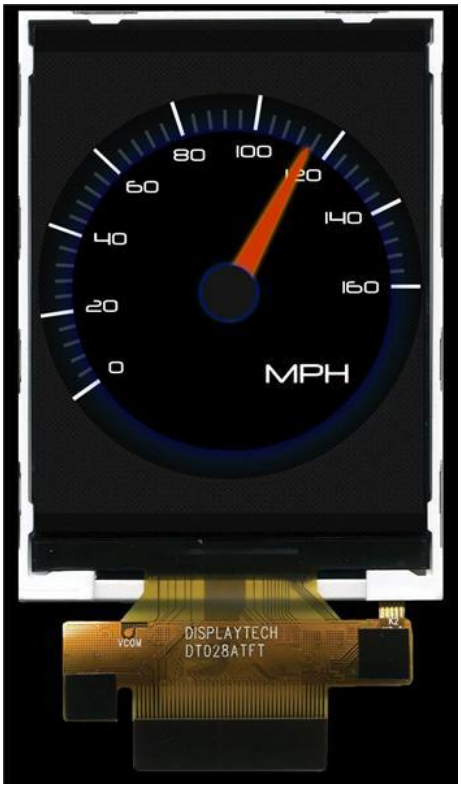


Obrázek 32 – Vývojový kit – Kompas[11]

4.4 GPS modul – L80

Obrázek 33 – GPS kit – L80

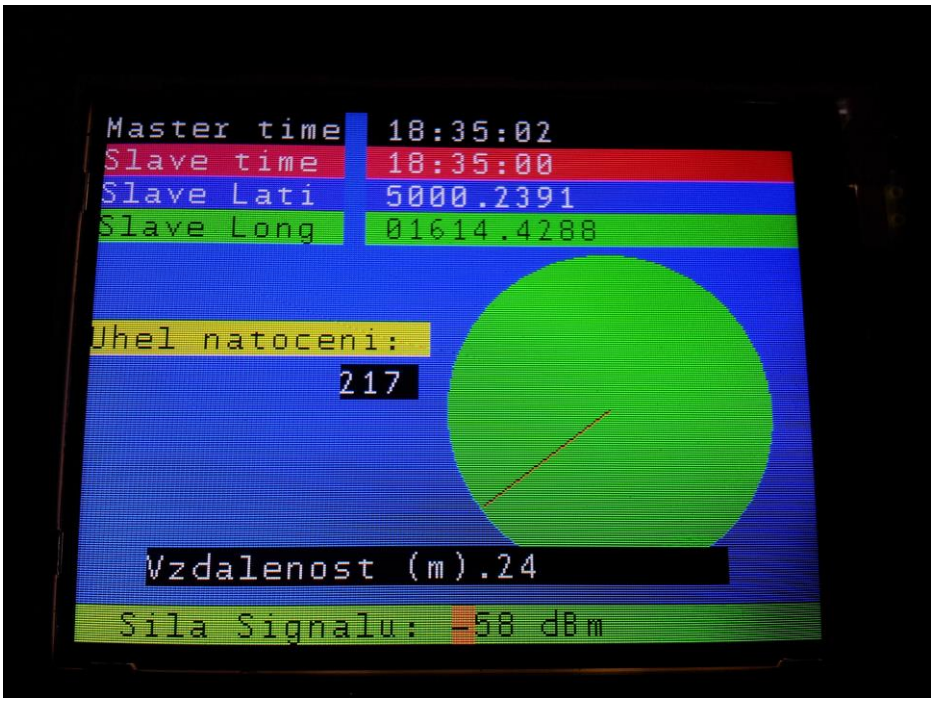
4.5 TFT – LCD Displej



Obrázek 34 – TFT displej [12]

Výpis údajů na displeji obsahuje:

- Hodiny z Master GPS
- Hodiny ze Slave GPS
- Zeměpisnou šířku i délku ze Slave zařízení
- Úhel natočení ke Slave přístroji a šipka, která na něj ukazuje
- Vzdálenost mezi body v metrech
- Úroveň signálu RSSI v jednotkách dBm



Obrázek 35– Vypsaná data na displeji

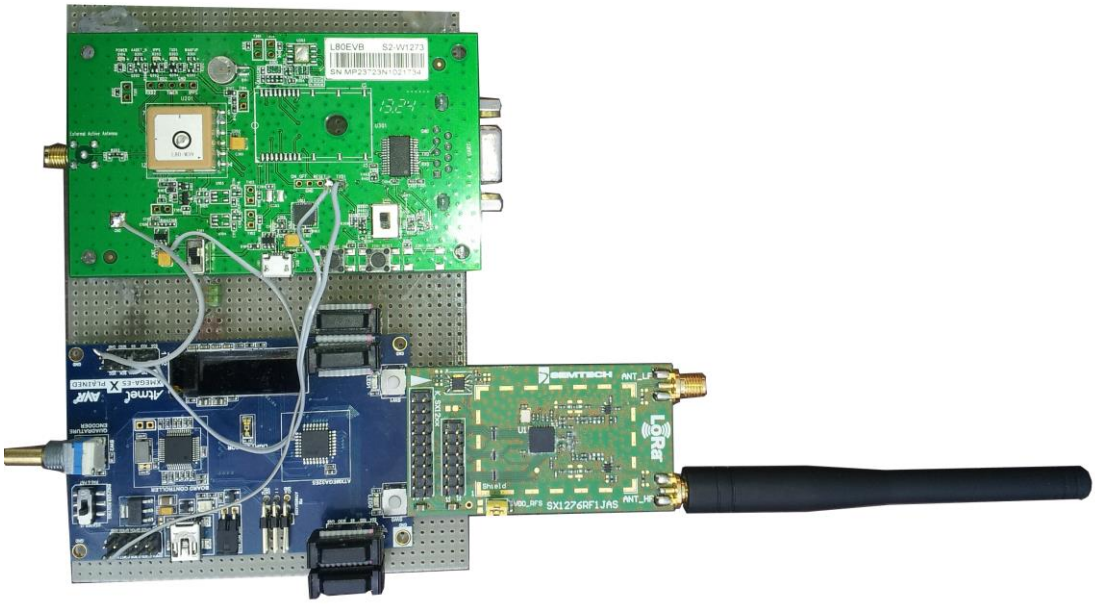
4.6 RF - Semtech SX1276 Kit

Obrázek 36 – SX1276 RF kit

4.7 Master zařízení

Obrázek 37 – Master zařízení

4.8 Slave zařízení



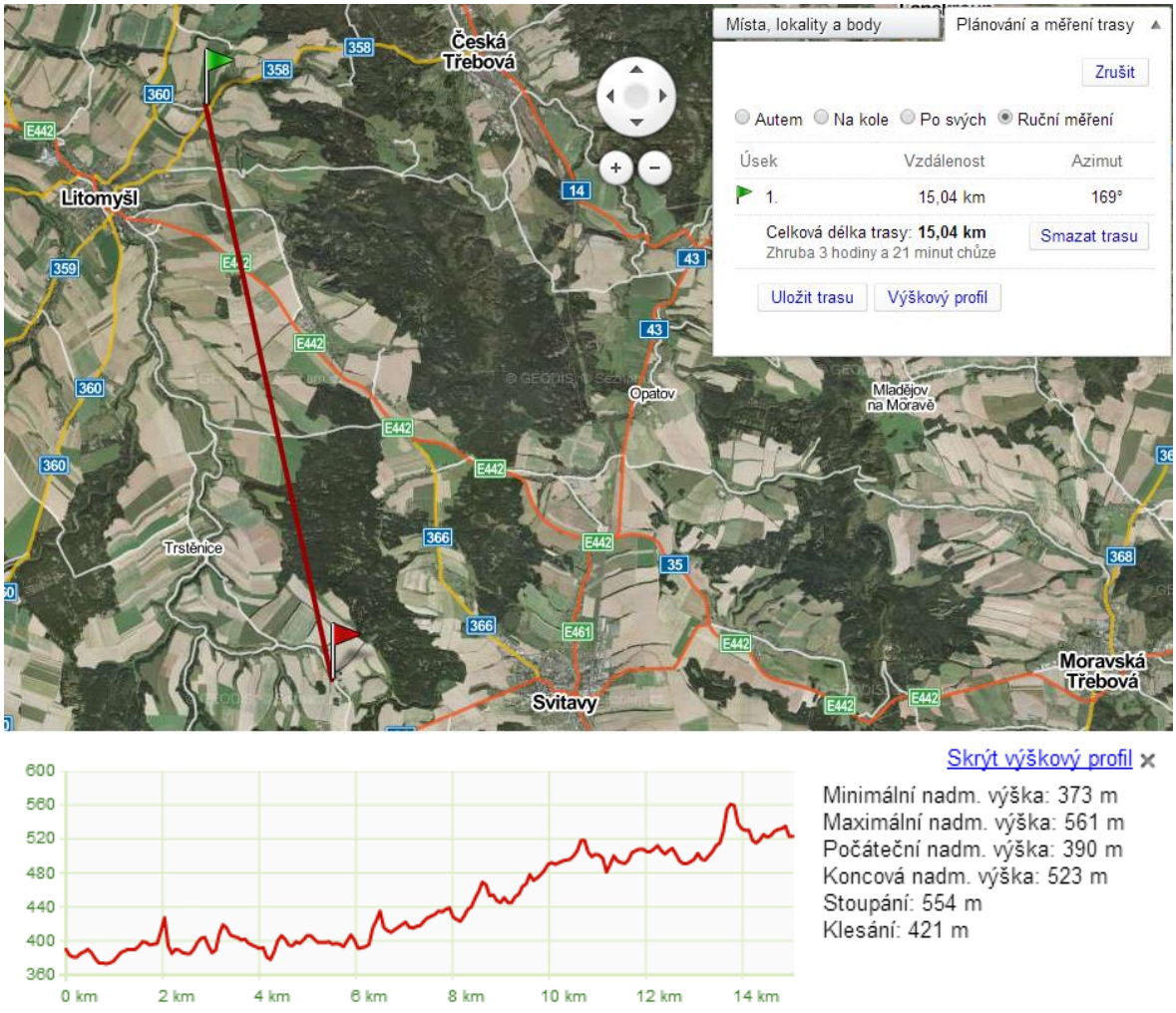
Obrázek 38 – Slave zařízení

5 Závěr

Cílem práce bylo sestrojit zařízení, které umožňuje sledovat daný objekt na vzdálenost až 5 km. Prozatím je přístroj postaven na kitech a není tedy možnost ho nějak vtěsnat nebo zabudovat do psího obojku nebo do uživatelsky pohodlné krabičky. Zařízení ovšem funguje a to na vzdálenost až 15 km.

Ačkoli se dá zařízení celkově zlepšovat, provedl jsem základní testy funkčnosti v terénu. Výpočet vzdálenosti pomocí elipsoidu WGS 84 je velmi přesný. Pro ověření přesnosti výpočtu vzdálenosti jsem umístil Master i Slave zařízení na fixní souřadnice a výsledek ukazovaný na Master zařízení se lišil v porovnání s výsledkem naměřeným na www.mapy.cz asi o 10 metrů.

V budoucnu se bude implementovat bezdrátový přenos pomocí GSM, tudíž bude vzdálenost sledování téměř neomezená, ovšem na úkor zpoplatnění přenosu v podobě uživatelského tarifu u mobilního operátora. Další výhodou GSM bude možnost připojení k internetu a jeho využití k získání almanachu pro rychlejší obdržení aktuální pozice GPS.



Obrázek 39– Zobrazení testovaných souřadnic na mapě [13]

6 Literatura

[1] *Mobilmania* [online]. 2014 [cit. 2014-04-28]. Dostupné z: http://www.mobilmania.cz/jak-urcit-polohu-mobilniho-telefonu/a-1107567/default.aspx

[2] GPS L80. *Quectel* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.quectel.com/product/prodetail.aspx?id=62

[3] SX1276. *Semtech* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.semtech.com/wireless-rf/rf-transceivers/sx1276/

[4] LSM303DLHC. *ST* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.st.com/web/en/resource/technical/document/datasheet/DM00027543.pdf

[5] EPARO. *Vyuka UPCE* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://vyuka.eparo.cz/isame/orientace.pdf

[6] Application note. *Pololu* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.pololu.com/file/download/LSM303DLH-compass-app-note.pdf?file_id=0J434

[7] Atmel. *Xmega32e5* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.atmel.com/Images/doc8077.pdf

[8] FreeRTOS. *FreeRTOS* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.freertos.org/

[9] Atmel Xmega E5 Explained. *Atmel* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.atmel.no/webdoc/xmegae5xplained/xmegae5xplained.generalinfo.html

[10] Atmel SAM4S. *Atmel* [online]. 2014 [cit. 2014-04-24]. Dostupné z: http://www.element14.com/community/servlet/JiveServlet/showImage/159063/ATSAM4S-XPRO_1.PNG

[11] ST. *STEVAL* [online]. 2014 [cit. 2014-04-24]. Dostupné z: http://www.st.com/st-web-ui/static/active/en/fragment/product_related/rpn_information/board_photo/image_steval-mki124v1.jpg

[12] TFT. In: *DTO28ATFT* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://cz.mouser.com/images/displaytechltd/lrg/DT028ATFT.JPG

[13] *Mapy* [online]. 2014 [cit. 2014-05-04]. Dostupné z: http://www.mapy.cz/