F3 Fakulta elektrotechnická
Katedra elektromagnetického pole

Bakalářská práce

Systém družicové navigace Galileo a algoritmy určení polohy

The Galileo Satellite Navigation System and Position Determination Algorithms

Tomáš Vaňkát

Vedoucí: Prof. Ing. František Vejražka, CSc.
Studijní program: Komunikace, multimédia a elektronika
Obor: Komunikační technika

Praha, Květen 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vaňkát** Jméno: **Tomáš** Osobní číslo: **457162**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektromagnetického pole**

Studijní program: **Komunikace, multimédia a elektronika**

Studijní obor: **Komunikační technika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém družicové navigace Galileo a algoritmy určení polohy

Název bakalářské práce anglicky:

The Galileo Satellite Navigation System and Position Determination Algorithms

Pokyny pro vypracování:

Prostudujte dokumentaci k systému Galileo a zhodnoťte možnost dekódování navigační zprávy a získání efemerid družic. Navrhněte algoritmus určení polohy uživatele, alespoň ve formě vývojového diagramu. Ověřte ho na simulovaných, příp. i na reálných datech.
Písemná zpráva: do 50 stran, příp. pravidelné dílčí zprávy.

Seznam doporučené literatury:

[1] ICD dokument Galileo, vydání 2014.
[2] Hrdina, Pánek, Vejražka: Skriptum Rádiové určování polohy. GPS.
[3] Kovář, P.: Družicová navigace, Praha, ČVUT, 2016.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. František Vejražka, CSc., katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2017** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce:
do konce letního semestru 2018/2019

prof. Ing. František Vejražka, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____
Datum převzetí zadání Podpis studenta

Čestné prohlášení

„Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.“

V Praze, 17. května 2018

.....
Tomáš Vaňkát

Poděkování

Děkuji zejména panu profesorovi Ing. Františkovi Vejražkovi, CSc., za celkové odborné vedení mé práce, podporu při tvorbě a konzultace k obsahu.

Velké poděkování patří také Ing. Václavu Navrátilovi, za konzultace ohledně problematiky výpočtů a korekcí v družicové navigaci. V neposlední řadě také děkuji Ing. Jiřímu Svatoňovi hlavně za umožnění ověření algoritmů jak na simulátoru, tak na vyvíjeném GNSS přijímači.

Abstrakt

Hlavním cílem této bakalářské práce je podrobně popsat, navrhnout a realizovat jednotlivé dílčí kroky a výpočty vedoucí od zpracování demodulovaného datového toku systému GNSS Galileo v rámci služby Open service, k výpočtu polohy přijímače v programovacím jazyku C. Těmito kroky je myšleno zejména dekódování navigační zprávy z demodulovaného signálu, extrakce dat z této zprávy a následné potřebné výpočty, například korekce systémového času, určení polohy družice a další algoritmy nezbytné ke zpracování dat.

Klíčová slova: GNSS, Galileo, Open Service, navigační zpráva, dekódování, určení polohy, korekce

Abstract

The main goal of this bachelor thesis is to devise, implement and thoroughly describe individual steps and calculations used in the processing of the data flow in GNSS Galileo Open Service which led to determining the position of the receiver in the programming language C. The main steps and calculations were the decoding of a navigation message from a demodulated signal, data extraction from the message and follow-up calculations such as time correction, determination of the satellite position and other algorithms necessary for data processing.

Keywords: GNSS, Galileo, Open Service, navigation message, decoding, positioning, correction

Obsah

Obrázky

Tabulky

Seznam zkratek

BGD	Broadcast Group Delay
C/NAV	Commercial Navigation Message
CRC	Cyclic Redundancy Check
EC	European Commission
ECEF	Earth Centred Earth Fixed
ECI	Earth Centred Inertial
ESA	European Space Agency
EU	Evropská unie
F/NAV	Freely accessible Navigation Message
FEC	Forward Error Correction
GLONASS	Global'naya Navigatsionnaya Sputnikovaya Sistema
GNSS	Global navigation satellite system
GPS	Global Positioning System
GPSt	GPS time
GSA	European GNSS Agency
GST	Galileo System Time
HD	Hamming distance
I/NAV	Integrity Navigation Message
ICD	Interface control document
int	integer
IOD	Issue of Data
IRNSS	Indian Regional Navigation Satellite System
LLA	Latitude, Longitude, Altitude
LSB	Least significant bit
MODIP	Modified dip latitude
MSB	Most significant bit
NAVSTAR GPS	Navigation Signal Timing and Ranging GPS
OS	Open Service
PRN	Pseudo-Random Noise
PRS	Public Regulated Service
PT	Page type
QZSS	Quasi-Zenith Satellite System
SAR	Search and Rescue
SISA	Signal-in-Space Accuracy
SVID	Space Vehicle IDentifier

TAI Mezinárodní atomový čas
TEC Total electron content
TOW Time Of Week
uint unsigned integer
UTC Universal Coordinate Time
WN Week Number
WT Word Type
XOR Exclusive or

Kapitola 1

Úvod

Algoritmy vytvořené v této bakalářské práce jsou určeny pro přijímač GNSS signálů, proto byl zvolen programovací jazyk C. V částech věnovaných právě těmto algoritmům je nejprve uveden princip metody a následně vysvětlena realizace. Zdrojové kódy algoritmů, které jsou uvedeny v příloze B, jsou určeny pro signál E1-B. Postupy pro ostatní signály služby OS jsou stejné nebo velmi podobné. Služba na signálu E5b se liší zejména uspořádáním subrámce, na E5a se liší jak délkou bloku dat (stránky), tak uspořádáním dat v navigační zprávě i jejím obsahem, ovšem principy zabezpečení přenosu i výpočtů jsou obdobné. Algoritmy byly otestovány post processingově na datech získaných jak z generátoru GNSS signálů, tak z reálného signálu družic.

Funkce zajišťující synchronizaci, dekódování, extrakci a uložení dat z navigační zprávy jsou volány v rámci řídící funkce pro zpracování dat *„process_GALILEO_data“*. Ta má jako vstupní parametry *SVID* a ukazatel na *buffer*. Proměnná *SVID* je číselné označení družice, získané ze znalosti rozdílných kódu jednotlivých družic. *Buffer* je posuvný registr o rozměru 16 x *uint*64, lze ho také interpretovat jako dvojrozměrné pole $16x64$ bitů, do kterého se vkládají bity demodulované navigační zprávy. Před vložením nového bitu se celý registr posune o jednu pozici „doleva“ (směrem k MSB) a na pozici uchovávající LSB (indexy [0][0]) se vloží nově demodulovaný bit. Výše uvedená řídící funkce je spuštěna vždy po přidání bitu do bufferu v rámci hlavního programu přijímače.

Výpočtové funkce jsou volány v rámci hlavní funkce výpočtů *„user_pos“*. Jde zejména o algoritmus určení polohy družice, korekce času, korigování šíření signálu v průchodu ionosférou, troposférou a výpočet polohy přijímače.

Všechny parametry a výsledky výpočtů, které musí být uchovány po delší dobu pro další využití, jsou definovány v rámci hlavičkového souboru, pojmenovaného „process_GALILEO_data.h“, stejně jako hlavičky všech funkcí. Zdrojové kódy jsou uloženy v souboru „process_GALILEO_data.c“.

Kapitola 2

Metody určování polohy

2.1 Dálkoměrná

Nejčastěji používaná metoda určení polohy v družicových navigačních systémech. Spočívá v měření doby šíření signálu τ_i od vysílače k uživateli. Ze znalosti tohoto časového úseku a rychlosti šířená signálu c (rychlost světla) snadno spočteme vzdálenost d_i přijímače od uživatele z následujícího vztahu:

$$d_i = c \cdot \tau_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \tag{2.1}$$

kde x_i, y_i, z_i jsou souřadnice družice a x, y, z souřadnice uživatele.[2] Tato vzdálenost vytyčuje množinu bodů v prostoru, tvořící povrch koule, kde se může uživatel nacházet vůči družici. Z toho je patrné, že je třeba znát vzdálenost od více, konkrétně tří, družic a jejich přesnou polohu, aby bylo možné určit jeden bod. Ve skutečnosti průsečíky tří koulí vytyčují dva body v prostoru, z nichž jeden je blíže Zemi. Pokud se má uživatel nacházet na povrchu Země, volí se právě bod blíže povrhu.[3] Polohu družice lze spočítat z dat, která vysílá. Tento postup by byl dostačující, pokud by časové základny družic a uživatele byly synchronní, hodiny uživatele jsou ovšem zpravidla posunuty o neznámý časový interval Δt, který lze vyjádřit jako vzdálenost $b = c \cdot \Delta t$. Z tohoto důvodu je třeba měřit zpoždění signálu od 4 družic a následně řešit čtyři rovnice pro čtyři neznámé. Měření se provádí zpravidla pomocí generování kopie přijímaného dálkoměrného PRN kódu, oba kódy se zesynchronizují a měří se pouze posun τ_{mi} začátku kopie kódu s počátkem časové základny přijímače. Tento časový posun, přepočítaný na vzdálenost D_i se označuje jako pseudovzdálenost.[2] Výše uvedenou rovnici 2.1 tak vyjádříme v konečném tvaru:

$$d_i = D_i + b = c(\tau_{mi} + \Delta t) = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + b \tag{2.2}$$

2.2 Dopplerovská

Tato metoda využívá Dopplerova jevu. Tento jev spočívá v rozdílnosti frekvence signálu vyslaného vysílačem f_v a frekvence téhož signálu přijatého přijímačem f_p za předpokladu, že se vysílač a přijímač oproti sobě pohybují nenulovou rychlostí a mění tak vzájemnou vzdálenost. Signálem jsou přenášeny časové značky v časech t_i, t_{i+1}, $t_{i+2}\ldots$ s pevným intervalem $T = t_{i+1} - t_i$. V přijímači je signál směšován se signálem pevného místního oscilátoru o frekvenci f_0. Následně se počítají periody

výstupního signály ze směšovače v průběhu intervalu T. Jinak řečeno, měří se změna fáze přijatého signálu mezi dvěma sousedními časovými značkami. Pokud spočítáme polohu vysílací družice (x_i, y_i, z_i) z přenášených parametrů v okamžiku vyslání časové značky a měření period N_i provedeme alespoň třikrát, můžeme následující rovnici 2.3 vyjádřit jako soustavu tří rovnic o třech neznámých.

$$N_i = T(f_0 - f_v) + \frac{f_0}{c}\left(\sqrt{(x_{i+1}-x)^2 + (y_{i+1}-y)^2 + (z_{i+1}-z)^2} - \right. \\ \left. - \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}\,\right) \tag{2.3}$$

Kde x, y, z jsou souřadnice uživatele a c je rychlost světla.[2]

2.3 Interferometrická

Metoda spočívá v diferenciálním měření signálu od jedné družice pomocí dvou antén, umístěných na základně přijímače se vzájemnou vzdáleností d. Jedna možnost je měření zdánlivých vzdáleností jednotlivých antén od družice a následné spočtení úhlu, mezi rovinou základny a spojnicí středu základny s družicí. Zjištěním tří takovýchto úhlů a znalosti souřadnic družice lze zjistit orientaci základny v prostoru. Druhá možnost je meření rozdílů fáze nosné vlny signálu přijímaného těmito dvěma anténami, ovšem počet celých period je třeba určit jinou metodou.[2]

2.4 Úhloměrná

Starší a málo přesná metoda, vyžadující směrové antény. Těmito anténami se měří elevační úhel družice vzhledem k referenční rovině (povrch Země, hladina). Z hodnoty tohoto úhlu určíme kužel s vrcholem v místě družice. Toto měření opakujeme několikrát v čase, nebo k více družicím. Z průsečíku kuželů a referenční hladiny určíme polohu uživatele.[2]

Kapitola 3

Družicové navigační systémy

Družicových navigačních systému v dnešní době existuje několik, některé jsou již dlouho zaběhlé a využívané (GPS, GLONASS), jiné jsou funkční z části nebo stále ve vývoji (Galileo, BeiDou). Tyto družicové systémy se dají rozdělit podle území které pokrývají na globální (Galileo, GPS..., viz níže) a regionální (IRNSS, QZSS...). Regionální pokrývají jen určitou část povrchu Země (např. Indie, Japonsko a okolí). Regionální systémy často pracují v koordinaci s některým globálním systémem a zlepšují tak jeho parametry v dané oblasti.

3.1 GPS

NAVigation Signal Timing And Ranging Global Positioning System (NAVSTAR GPS) byl prvním fungujícím dálkoměrným globálním družicovým navigačním systémem, vytvořeným původně pro účely americké armády. Počáteční operační dostupnost byla ohlášena v roce 1993 a plně funkčním se stal v roce 1995, kdy bylo dosaženo kompletního počtu družic na oběžné dráze pro určení polohy kdekoliv na Zemi. Později byl zpřístupněn i veřejnosti. Kosmický segment tvoří 24 družic obíhajících kolem Země po 6 různých drahách se sklonem 55° k rovníku, ve výšce přibližně 20200km a dobou oběhu 11h a 58min. K rozlišení jednotlivých družic používá kódové dělení, rozdílné kódy jsou využity zároveň jako dálkoměrné.[2][4][5] Momentálně je na oběžné dráze dostupných 31 družic.[6]

3.2 GLONASS

Tento ruský systém sloužil, obdobně jako GPS, nejprve pouze vojenskému účelu, později se jeho využití rozšířilo i do civilní sféry. Plnou konstelaci tvoří 24 družic, umístěných na 3 různých oběžných drahách vzájemně svírajících úhel 120° a s inklinací k rovníku 64,8°. Družice se nacházejí ve výšce přibližně 19100km nad povrchem Země a doba oběhu je necelých 11 hodin a 15 minut. GLONASS používá pro rozlišení jednotlivých družic frekvenční dělení dálkoměrného kódu (narozdíl od GPS a Galilea využívajících dělení kódové).[7][3]

3.3 BeiDou

Čína vyvinula systém Beidou (Beidou 1) původně pouze jako regionální navigační systém s geostacionárními družicemi. Později koncept pozměnila a nyní rozšiřuje systém tak, aby se z něj stal plnohodnotný globání navigační systém (Beidou 2 resp. Compass). Měl by být tvořen celkem 35 družicemi a plně funkční do roku 2020.[7][3][8]

3.4 Galileo

Galileo je jediný čistě civilní družicový navigační systém, řízený organizacemi EU, a to Evropskou kosmickou agenturou (ESA), Evropskou komisí (EC) a Evropskou GNSS agenturou (GSA). Má zajistit nezávislost EU v oblasti družicové navigace na mimoevropských vojenských systémech (GPS, GLONASS). První družice byly vyneseny na oběžnou dráhu v roce 2011, plné konstelace má být dosaženo v roce 2020.[9] Plnou konstelaci má tvořit celkem 30 družic včetně záložních. Průměrná délka hlavní poloosy oběžné dráhy je 29600km, družice tak obíhají ve výšce přibližně 23222km nad Zemí po třech oběžných rovinách se sklonem 56° k rovníku. K rozlišení družic je použito, stejně jako u GPS, kódové dělení.[1]

Kapitola 4

Souřadné systémy v GNSS

V družicových navigačních systémech se využívá hned několik souřadných soustav, některé jsou inerciální, jiné neinerciální rotační. Pro jejich výhody i nevýhody se používají systémy z obou skupin. Pro potřeby této práce si vystačíme se dvěmi vyjádřeními neinerciální soustavy ECEF a jednou inericální soustavou ECI.

4.1 ECI

Earth-Centered Inertial (ECI) je inerciální souřadnicová soustava s počátkem v gravitačním středu Země. Směr jednotlivých os x, y, z je fixován a respektuje polohu hvězd. Rovina vytyčená osami x a y se shoduje s rovinou zemského rovníku, kdy osa x je spjata s bodem na nebeské sféře, nejčastěji s tzv. jarním bodem. Osa z je kolmá na rovinu xy a směřuje do bodu severního pólu. Osa y je volena tak, aby systém tvořil pravotočivou soustavu.[3] V inerciálních soustavách platí Einsteinova teorie relativy, takže je v nich známá rychlost šíření signálu, stejně tak v nich platí Newtonovy pohybové zákony. Tyto vlastnosti jsou nezbytné pro jednotlivé výpočty, například Newtonovy pohybové zákony jsou potřeba pro výpočet dráhy a polohy družice, rychlost šíření signálu zase pro výpočet polohy uživatele.[9]

4.2 ECEF

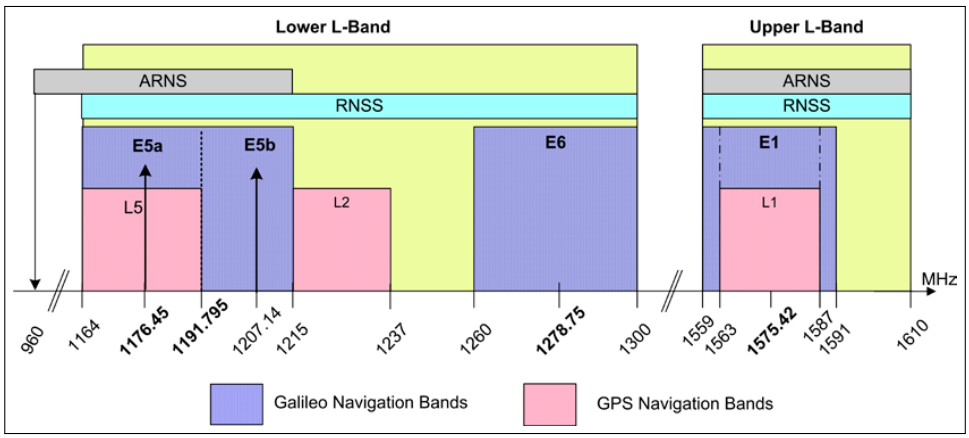
Soustava ECEF (earth-centered, earth-fixed) je neinerciální souřadná soustava pevně spjatá se Zemí. Její počátek je v gravitačním středu Země a otáčí se s společně s planetou. Polohu v této soustavě určujeme buď běžnými geodetickými souřadnicemi tzv. LLA, z anglického Latitude, Longtitude, Altitude (zeměpisná šířka φ, zeměpisná délka λ a výška h) nebo kartézskými souřadnicemi x, y, z, kdy osa z prochází zemským pólem, osa x prochází průsečíkem rovníku a referenčního poledníku, osa y je volena tak, aby tvořila pravotočivý souřadný systém.[3][9]

Kapitola 5

GNSS Galileo

5.1 Specifikace služeb

Galileo využívá stejné frekvenční pásmo jako ostatní družicové navigační systémy. Jednotlivé nosné vlny a kanály jsou znázorněny na obrázku 5.1 níže. Nosné E5a a E5b jsou součástí frekvenčního spektra signálu E5 (1191,795MHz). Podrobnější informace o jednotlivých signálech a jejich modulacích jsou uvedeny v ICD [1].



Obrázek 5.1: Frekvenční plán GALILEO a GPS[1]

5.1.1 Open Service

Základní volně dostupná služba pro běžné uživatele. Slouží zejména k určování polohy a času (automobilová navigace, mobilní telefony...). Data této služby jsou přenášena pomocí dvou navigačních zpráv: I/NAV na signálech E1 a E5b, F/NAV na signálu E5a (viz níže).[1] V rámci OS tak lze pro měření použít jak jednofrekvenční příjimač, tak i dvoufrekvenční pro dosažení přesnějšího určení polohy a eliminaci některých chyb.

5.1.2 Commercial Service

Služba zaměřená na komerční aplikace, vyžadující vyšší přesnost a výkonnost. K OS poskytuje další signál s dalšími informacemi, který bude kódovaný. Přístup k ní bude možný jen s klíčem od poskytovatele služby po zaplacení poplatku.[10] Navigační zpráva C/NAV tvořící tuto doplňkovou službu je šířena pomocí frekvenčního pásma E6.[1]

5.1.3 Public Regulated Service

Veřejně regulovaná služba (PRS) je šifrovaná a určená pouze pro státem autorizované uživatele (typicky pro bezpečnostní složky státu). Měla by být dostupná za všech okolností a velmi dobře zabezpečena proti zneužití, výpadkům nebo případným útokům.[10]

5.1.4 Search And Rescue service

Služba SAR poskytuje možnost aktivní oboustranné komunikace mezi uživatelem a družicemi v rámci systému COSPAS-SARSAT. Slouží k nouzové lokalizaci plavidel, letadel nebo uživatelů, kteří vyšlou záchranný signál z vysílače určeného k tomuto účelu.[10]

5.2 Specifikace dat

Přenášená data lze rozdělit do několika hlavních skupin:

- **Efemeridy** vysílající družice, včetně korekčních hodnot keplerovských parametrů. Dále almanach s méně přesnými efemeridy ostatních družic.
- **Typ datového slova**, neboli word type (WT), jednoznačně určující svou hodnotou jaké parametry a na jakých pozicích se v daných datech nacházejí. Pro nominální stránky I/NAV nabývá hodnot 0 až 10, hodnota 63 indikuje tzv. „Dummy message“, kterou se nahrazuje nominální stránka v případě nevalidních dat. Pokud má jinou hodnotu, tak se jedná o zatím nevyužité, rezervované datové místo nebo neopravenou chybu v přenosu.[1] Navigační zpráva F/NAV používá obdobný parametr, označovaný jako page type.
- **Korekční parametry.** Časové, sloužící ke stanovení přesného času vyslání signálu. Skupinové zpoždění výstupních obvodů družice BGD. Ionosférické, potřebné k výpočtu modelu ionosférického zpoždění signálu.
- **Časové parametry** určující aktuální systémový čas GST a parametry nutné pro přepočet na jiný časový systém (UTC, GPSt).
- **Integrita a stav systému** zahrnující informace o validitě přenášených dat, označení souboru dat (Issue of data), stavu vysílaného signálu (v pořádku, mimo provoz, testování…) a predikci přesnosti signálu (SISA).

Všechny parametry mají definovaný počet bitů a jednotku. Jsou přenášeny jako celočíselná hodnota a z toho důvodu má téměř každý parametr definované své měřítko, kterým je třeba ho vynásobit pro získání skutečné hodnoty parametru.[1] Typicky jde o činitel $60 \cdot N$ v případě časových parametrů nebo činitel 2^{-N} (kde N je celé číslo) v případě neceločíselných parametrů. Parametry, které jsou přenášeny bez přepočtení hodnoty, mají činitel 1 nebo nedefinovaný. Jednotlivé parametry jsou popsány dále v rámci algoritmu, který s nimi pracuje.

5.3 Vysílané navigační zprávy OS

Obě níže popsané navigační zprávy jsou vysílány v rámci služby OS. Využívají totožné metody zabezpečení přenosu (viz kapitola 6), ovšem liší se délka jednotlivých bloků dat, rychlost přenosu a samozřejmě vysílací signál. Daty přenášenými v rámci služeb s omezeným přístupem se tato práce nezabývá.

5.3.1 F/NAV

Freely accessible Navigation Message je základní navigační zpráva obsahující pouze navigační data potřebná k určení času a polohy, neposkytuje informace o integritě celého systému. Je vysílána na signálu E5a-I rychlostí 50 *symb*/*s*. Jedna stránka zprávy (samostatný blok dat) je tvořena 244 bity dat (ty se zakódují do 488 symbolů, viz sekce 6.2) a synchronizačním slovem o délce 12 bitů. Jedna stránka se tak vysílá po dobu 10 sekund. Pět stránek tvoří subrámec a 12 subrámců tvoří jeden rámec. V každém subrámci jsou přenášena kompletní navigační data dané družice (efemeridy, časové a ionosférické korekce ...) a část almanachu s méně přesnými efemeridami dalších družic. V následující tabulce 5.1 je znázorněna struktura jedné stránky navigační zprávy.

12	488				symbolů
12	244				bitů
Synch. slovo	Page Type	Data	CRC	Tail	Obsah
12	6	208	24	6	bitů

Tabulka 5.1: Struktura stránky F/NAV[1]

Page type (PT) nabývá hodnot od 1 do 6 a slouží k jednoznačné identifikaci dat obsažených na dané stránce navigační zprávy.[1] Ostatní bloky uvedené v tabulce jsou popsány dále pro navigační zprávu I/NAV, jejich účel je stejný pro obě tyto zprávy. Rozdílné je rozložení přenášených parametrů na stránkách. Data obsažená na jednotlivých stránkách jsou znázorněna v následující tabulce 5.2. Podrobné umístění jednotlivých parametrů lze nalézt v ICD [1, str. 30]. V každém subrámci jsou přenášeny vždy stránky typu 1 až 4, stránky 5 a 6 se střídají po subrámci.

Page Type	Obsažená data
1	Časové a ionosférické korekce, GST, SVID, BGD, SISA, validita dat a signálu
2	Efemeridy, GST
3	Efemeridy, GST
4	Efemeridy, část GST, přepočtové paramatry GST-UTC a GST-GPSt
5	Almanach
6	Almanach

Tabulka 5.2: Datový obsah stránek F/NAV

5.3.2 I/NAV

Integrity Navigation Message obsahuje jak navigační data potřebná k určení polohy uživatele a času (efemeridy družic...), tak informace o integritě systému a v případě signálu E1-B i SAR data. Šířena je pomocí signálů E5b-I a E1-B rychlostí 250 $symb/s$. Jedna stránka navigační zprávy se rozděluje na dvě stejně velké části, označované jako „sudá“ (Even) a „lichá“ (Odd). Obě tyto části jsou kódovány a vysílány samostatně a na začátku každé této části je zařazeno deseti bitové synchronizační slovo. Jedna část stránky je složena ze 120 bitů, které se zakódují do 240 symbolů, před tento blok se zařadí synchronizační slovo, které se nekóduje, celkem tedy 250 symbolů.[1][11]

10	240									symbolů
10	120									bitů
Synch. slovo	Odd	Page Type	Data j (2/2)	Reserved 1	SAR	Spare	CRC_j	Reserved 2	Tail	Obsah
10	1	1	16	40	22	2	24	8	6	bitů
Synch. slovo	Even	Page Type	Data k (1/2)						Tail	Obsah
10	1	1	16	40	22	2	24	8	6	bitů

Tabulka 5.3: Struktura stránky I/NAV na signálu E1-B[1]

Na signálu E5b jsou pole SAR a Spare nahrazena širším polem Reserved 1. Ve stejný okamžik se na každém z obou signálů vysílá jiná stránka navigační zprávy. Zároveň se v ten samý okamžik na jednom signálu vysílá lichá část stránky a na druhém signálu sudá část. Stránky jsou v subrámci také jinak poskládány pro oba signály (viz ICD [1, str. 34]). Jeden rámec je tvořen 24 subrámci a subrámec 15 kompletními stránkami.

Pole Even(=0)/Odd(=1) určuje o jakou část stránky se jedná. Page type indikuje v případě „0“ nominální stránku, v případě „1“ výstražnou stránku. SAR data jsou

blíže specifikována v [1, str. 40]. Pole Reserved a Spare jsou zatím nedefinována a jsou určena pro budoucí využití. Synchronizační slovo (sekce 6.1), Data (sekce 5.2), CRC (sekce 6.4) a Tail (sekce 6.2) jsou popsány v jiných částech této práce. Níže uvedená tabulka 5.4 ilustruje data obsažená v jednotlivých datových slovech WT. Rozložení konkrétních parametrů je uvedeno v [1, str. 37].

Word Type	Obsažená data
1	Efemeridy
2	Efemeridy
3	Efemeridy, SISA
4	Efemeridy, SVID, časové korekční parametry
5	Ionosférické korekce, GST, BGD, validita signálů a dat
6	Parametry konverze GST-UTC, část GST
7	Almanach
8	Almanach
9	Almanach
10	Almanach, parametry přepočtu GST-GPSt
0	GST, validita GST

Tabulka 5.4: Datový obsah stránek I/NAV

5.4 Algoritmy pro třídění a ukládání dat

Po dekódování kompletní stránky navigační zprávy (v případě I/NAV tedy obou polovin) a ověření, že v datech není chyba pomocí CRC, můžeme přistoupit ke čtení jednotlivých parametrů. Přečteme typ datového slova, díky tomu zjistíme přesně jaké parametry a v jakém pořadí se v datech nacházejí. Parametry jsou přesně definovány v ICD [1], takže víme, zda nabývají pouze kladných hodnot, nebo kladných i záporných hodnot a z kolika bitů se skládají. Pro každý vysílaný parametr je vytvořena celočíselná proměnná o co nejmenší velikosti, aby se do ní dal parametr uložit (např. pro parametry zabírající 12 bitů, je v programu vytvořena proměnná o délce 16 bitů). Pokud přenášený parametr nabývá pouze jedné polarity, ukládá se v programu do unsigned integer proměnných (celočíselné neznaménkové, např. uint8). Pokud nabývá dvojí polarity, ukládá se do signed integer proměnných (např. int16). Tyto proměnné jsou definovány v globálních strukturách v rámci hlavičkového souboru. Ukládání probíhá po jednotlivých bitech. Po přečtení každého bitu se zjistí jeho hodnota, v případě hodnoty bitu „1“ se inkrementuje proměnná o jedničku, v případě hodnoty bitu „0“ se proměnná nemění. Následně se provede bitový posun proměnné o jedno místo doleva a čte se další bit. V případě přenášeného parametru který nabývá obou polarit se nejprve přečte hodnota parametru bez znaménkového bitu (MSB) a následně se přečte hodnota znaménkového bitu (MSB) samostatně, vynásobí se číslem 2^{len-1}, kde len je délka parametru v bitech a odečte se od hodnoty parametru. Takto se převede znaménkové číslo z binární soustavy do soustavy desítkové.

V následující tabulce jsou znázorněné datové typy proměnných pro parametry přenášené ve word type 2:

WT=2	IOD_{nav}	t_{0e}	M_0	e	$\sqrt{A}$	Reserved	
6	10	14	32	32	32	2	délka v bitech
+	±	±	±	±	±	±	polarita
-	uint16	int32	int32	int32	int16	-	datový typ

Tabulka 5.5: Příklad datových typů proměnných pro parametry word type 2

Plnění jednotlivých proměnných přenášenými parametry zajištujě funkce *„sortWTX“*, kde X je hodnota WT. Vstupními parametry jsou identifikátor družice (SVID) a pole obsahující stránku s daty. Pro vykopírování hodnoty ze stránky navigační zprávy volá funkce *„sortWTX“* pro každou proměnnou funkci *„readvalue“*, jejími vstupními parametry jsou pořadí bitu na stránce, na kterém parametr začíná *begin*, jeho délka v bitech *len*, typ polarity *sign* a stránka dat *page*. Návratovou hodnotu tvoří právě přečtený parametr. Realizace čtení parametrů z bitů stránky navigační zprávy je znázorněno na následujícím úseku kódu.

```
readvalue(begin, len, page[], sign)
val=0;
pom=0;
beg=begin;
i=begin;
if (sign==1) //1 - Parametr nabývá obou polarit, 0 - pouze kladné
 then
    beg++;
    if (page[begin]==1) //Znaménkový bit = 1 -> záporná hodnota
     then
        pom = 2^(len−1);
    end
end
for ( i=beg ; i<(begin+len) ; i++ ) do
    val=val«1;
    if (i<114)//Po 1. pol. str. je 8 bitů bez dat (Tail, E/O, PT, viz tab. 5.3)
     then
        if (page[i]==1) then
            val++;
        end
    else
        if (page[i+8]==1) then
            val++;
        end
    end
end
return val-pom;
```

Kapitola 6

Zabezpečení přenosu

Každý datový přenos je vhodné určitým způsobem ošetřit, aby šlo jednotlivá data jednoznačně identifikovat, minimalizovat možnost chybného přenosu nebo zjistit, zda byla nějaká chyba do přenosu zanesena a případně ji opravit. K těmto účelům slouží například kódování, které přidává redundatní informaci, sloužící k opravě chyb při dekódování nebo přenášení hodnoty redundantního kontrolního součtu a další níže uvedené metody.

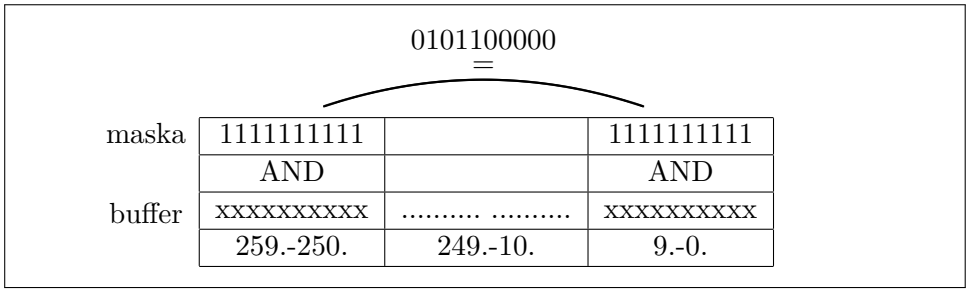
6.1 Synchronizace dat

Synchronizace datového toku navigační zprávy je zajištěna vkládáním přesně definovaného synchronizačního slova (viz tabulka 6.1) na začátek každého vysílaného zakódovaného bloku (stránky v případě F/NAV, sudé nebo liché části stránky v případě I/NAV). Toto synchronizační slovo není kódováno ani chráněno kontrolním součtem, případnou chybu v synchronizačním slově tak nelze opravit a nelze ho použít pro detekování začátku stránky. Pro oba typy navigační zprávy má synchronizační slovo jiný tvar i délku.[1]

Typ zprávy	Délka slova v bitech	Tvar slova
I/NAV	10	0101100000
F/NAV	12	101101110000

Tabulka 6.1: Struktura synchronizačního slova

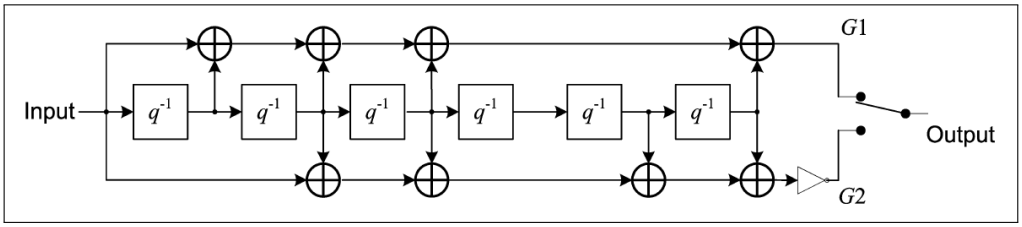
V programu se hledá synchronizační slovo po každém přidání nového demodulovaného bitu do *bufferu*. Ověří se, zda se poslední demodulované bity shodují se synchronizačním slovem. Pokud se shodují, ověří se, zda se shodné synchronizační slovo nachází i o X bitů v *bufferu* dříve (kde X je počet symbolů zakódovaného bloku/stránky navigační zprávy (250 pro I/NAV, 500 pro F/NAV)). Tímto dvojím ověřením se eliminuje možnost nalezení skupiny bitů, které se náhodně shodují se synchronizačním slovem. Pouze pár požadovaných bitů z bufferu získáme „vymaskováním“ (bitové vynásobení (operace *AND*) proměnné maskou stejné délky, která má jednotkové bity pouze na pozicích shodných s pozicemi bitů, které chceme získat). Je třeba testovat synchronizaci jak pro výše definované synchronizační slovo, tak v případě neshody i pro invertované synchronizační slovo. Taková situace může nastat pokud se demodulační obvody přijímače zachytí na signál s o 180° otočenou fází, protože použitá BOC modulace je symetrická. V důsledku toho se mohou demodulované úrovně signálu identifikovat opačně a demodulované bity mohou následně nabývat inverzních hodnot.



Obrázek 6.1: Hledání synchronizace v I/NAV

6.2 Kódování

Kódování zajišťuje konvoluční FEC kodér o omezující délce K=7, realizovaný posuvným registrem s kódovací rychlostí 1/2 (jeden vstupní bit se zakóduje do dvou výstupních symbolů). Omezující délka udává, kolik vstupních bitů ovlivňuje jeden výstupní symbol (v tomto případě tedy jeden výstupní symbol je ovlivněn 6 předchozími a 1 aktuálním vstupním bitem). Tento kodér může nabývat $2^6 = 64$ stavů (0 až 63) a výpočet lze vyjádřit dvěma generujícími polynomy $G1 = 171_8 = 1111001_2$ a $G2 = 133_8 = 1011011_2$. Výstup je tvořen výsledkem operace XOR (součet modulo 2) získaný pomocí $G1$ a po něm invertovaný výsledek operace XOR pomocí $G2$. Na konec každého kódovaného bloku dat se zařazuje Tail, což je posloupnost šesti nul, která má za úkol dostat kodér do výchozího nulového stavu (kodér tak při kódování každého bloku začíná i končí ve stavu 0).[1][3]



Obrázek 6.2: Schéma konvolučního kodéru [1]

Z výše uvedeného schématu 6.2 je patrné, že po zakódování bitu na vstupu (Input) se tento bit vsune na 1. pozici kodéru (levý krajní blok q^{-1}), hodnota z 1. pozice se posune na 2. a tak dále až do poslední pozice, která se jen přepíše a její předchozí hodnotu dále nevyužíváme.[12] Jelikož každý bit může nabývat jen dvou hodnot, na vstupu se může nacházet buď hodnota 1 nebo 0, kodér se tak může dostat s každým posunutím do dvou jasně definovaných následujících stavů (podle hodnoty na vstupu), stejně tak do každého stavu se dá dostat přesně ze dvou různých předcházejících.

Mějme například kodér ve stavu 15_{10}=001111_2 a na vstupu hodnotu 1, na tuto posloupnost sedmi bitů aplikujeme výše uvedené operace XOR s polynomem Gx, kde x=1 resp. x=2:

in	kodér		Gx	out
1	001111	$\oplus$	1111001	= 1
!(1	001111	$\oplus$	1011011)	= !0 = 1

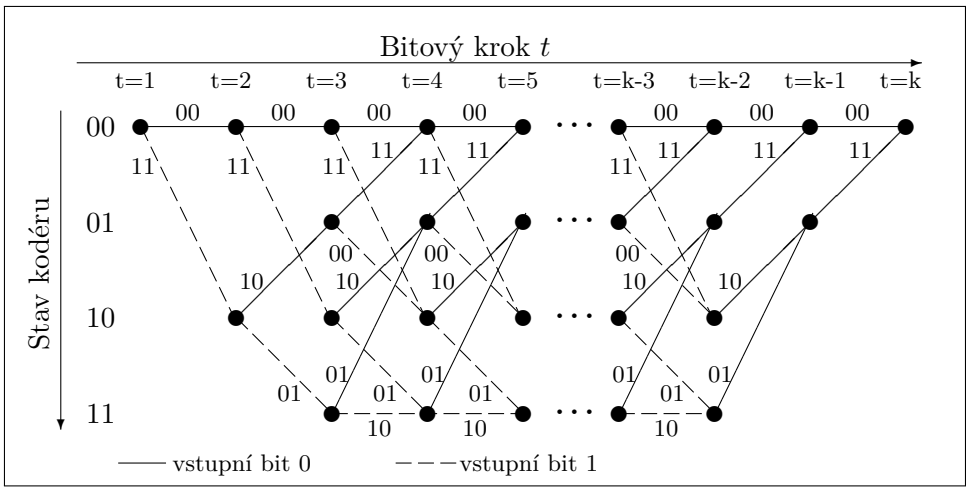
Tabulka 6.2: Příklad kódování jednoho bitu

Na výstupu dostaneme zakódovaný vstupní bit do dvojice symbolů 11. Poté se hodnoty v kodéru posunou a dostane se tak do dalšího stavu, v tomto případě do stavu $39_{10} = 100111_2$. V níže uvedené tabulce 6.3 jsou uvedeny všechny stavy kodéru (s_i), vyjádřené v desítkové soustavě, výstupní symboly z kodéru pro kódovaný bit = 1 (out_1), následující stav kodéru po kódování hodnoty 1 ($s_{1:i+1_{10}}$), výstupní symboly z kodéru pro kódovaný bit = 0 (out_0) a následující stav kodéru po kódování hodnoty 0 ($s_{0:i+1_{10}}$).

$s_{i_{[10]}}$	out_1	$s_{1:i+1_{[10]}}$	out_0	$s_{i+1_{[10]}}$		$s_{i_{[10]}}$	out_1	$s_{i+1_{[10]}}$	out_0	$s_{0:i+1_{[10]}}$
0	10	32	01	0		32	00	48	11	16
1	01	32	10	0		33	11	48	00	16
2	11	33	00	1		34	01	49	10	17
3	00	33	11	1		35	10	49	01	17
4	10	34	01	2		36	00	50	11	18
5	01	34	10	2		37	11	50	00	18
6	11	35	00	3		38	01	51	10	19
7	00	35	11	3		39	10	51	01	19
8	01	36	10	4		40	11	52	00	20
9	10	36	01	4		41	00	52	11	20
10	00	37	11	5		42	10	53	01	21
11	11	37	00	5		43	01	53	10	21
12	01	38	10	6		44	11	54	00	22
13	10	38	01	6		45	00	54	11	22
14	00	39	11	7		46	10	55	01	23
15	11	39	00	7		47	01	55	10	23
16	01	40	10	8		48	11	56	00	24
17	10	40	01	8		49	00	56	11	24
18	00	41	11	9		50	10	57	01	25
19	11	41	00	9		51	01	57	10	25
20	01	42	10	10		52	11	58	00	26
21	10	42	01	10		53	00	58	11	26
22	00	43	11	11		54	10	59	01	27
23	11	43	00	11		55	01	59	10	27
24	10	44	01	12		56	00	60	11	28
25	01	44	10	12		57	11	60	00	28
26	11	45	00	13		58	01	61	10	29
27	00	45	11	13		59	10	61	01	29
28	10	46	01	14		60	00	62	11	30
29	01	46	10	14		61	11	62	00	30
30	11	47	00	15		62	01	63	10	31
31	00	47	11	15		63	10	63	01	31

Tabulka 6.3: Stavy a výstupy kodéru systému Galileo

Celý postup kódování se všemi možnými vstupními datovými sekvencemi o délce k bitů je možné znázornit graficky, například v podobě mřížkového kódovacího diagramu (trellis diagram). Protože kodér znázorněný na obrázku 6.2 je příliš dlouhý pro názornou ukázku kvůli velkému počtu stavů kodéru, tak na následujícím obrázku 6.3 je uveden mřížkový diagram kodéru s omezující délkou K=3 (tedy se 4 stavy) a kódovací rychlostí 1/2. Na konec datové sekvence se zařazuje posloupnost $(K-1)$ nulových bitů, aby se kodér dostal opět do výchozího nulového stavu a usnadnilo se tak dekódování. U každé cesty mezi stavy uvedená dvojice symbolů znázorňuje výstup kodéru pro daný vstupní bit a výchozí stav.



Obrázek 6.3: Příklad kódovací mřížky kodéru

6.2.1 Popis dekódování

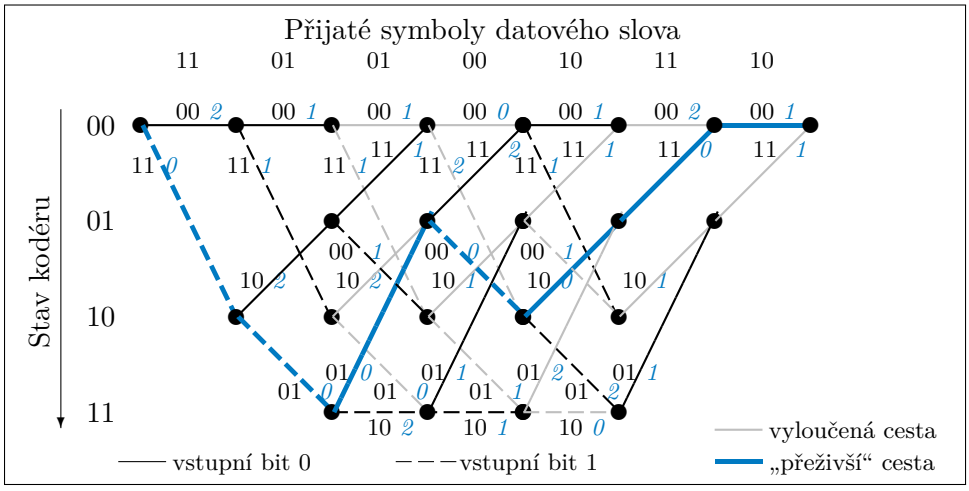
Aby bylo možné dále rozebrat postup dekódování, musí být zadefinována určitá metrika, na jejímž počítání je použité dekódování založeno. Jako tuto metriku použijeme Hammingovu vzdálenost (HD), která obecně udává počet rozdílných pozic, mezi dvěma vektory.[13] Pro naše potřeby HD označuje „nejmenší počet rozdílných bitů ve dvou stejně dlouhých kódových blocích.“[12] Například máme-li dvě následující kódová slova o délce 15 bitů:

101100101110010
111100110010110

vidíme, že se liší právě v 5 bitech. Jejich vzájemná Hammingova vzdálenost je tedy rovna 5.

Dekódování je zajištěno implementací Viterbiho algoritmu, jinak také dekódování s maximální pravděpodobností. Tento mechanismus předpokládá, že všechna možná platná kódová slova (zakódované datové sekvence) mohou být vysílána se stejnou pravděpodobností. Následně využívá znalosti kódové mřížky kodéru k nalezení takového kódového slova, které má nejmenší Hammingovu vzdálenost od slova přijatého.[12] Protože do každého uzlu kódové mřížky se dá dostat právě dvěmi cestami, spočteme Hammingovu vzdálenost obou cest od přijatého kódového slova a následně vybereme tu pravděpodobnější, tedy s nižší HD. Méně pravděpodobnou cestu můžeme vymazat, protože dál by obě cesty pokračovaly stejně. Jestli že nastane situace, kdy obě

cesty budou mít stejnou HD, tak je jedno kterou z nich vybereme, protože jsou obě stejně pravděpodobné a dále budou opět pokračovat stejnou cestou. Pokud takto projdeme celou mřížkou o délce k bitových kroků od prvního do posledního kroku a v každém kroku najdeme pravděpodobnější cestu do každého ze 64 stavů kodéru (uzlů), v posledním kroku nám zbude jediná cesta mřížkou, která vyjadřuje kódové slovo s nejmenší Hammingovou vzdáleností od kódového slova přijatého.[13][12] Tato „přeživší“ cesta musí nutně začínat i končit ve stavu 0. Na následujícím obrázku 6.4 je znázorněno přiřazení HD jednotlivým cestám mezi stavy v každém bitovém kroku na základě přijatého kódového slova a vybrání nejpravděpodobnější cesty v kódové mřížce stejného jednoduchého konvolučního kodéru popsaného výše. Určená Hammingova vzdálenost každého přechodu je uvedena kurzívou u dvojice symbolů znázorňující výstup kodéru. Zvýrazněna je cesta mřížkou s nejmenší HD od přijatého kódového slova, světle jsou označeny vyloučené přechody vedoucí do každého stavu.



Obrázek 6.4: Výběr nejpravděpodobnějšího průchodu mřížkou

Hammingova vzdálenost celé nejpravděpodobnější zvýrazněné cesty od přijatých symbolů je rovna 1, zatímco HD cesty vyloučené až v posledním kroku je rovna 4.

6.2.2 Realizace dekodéru

Z výše uvedeného popisu kódování (6.2.1) je patrné, že algoritmus musí být realizován dvěma cykly. První (vnější) cyklus prochází postupně po dvojicích přijaté symboly daného zakódovaného bloku dat (stránky navigační zprávy) a druhý (vnitřní) cyklus, vnořený do vnějšího, prochází přes všechny stavy kodéru. Pro dekódování bloku z navigační zprávy I/NAV je třeba vnější cyklus provést 120krát, protože tento blok obsahuje 240 symbolů (viz sekce 5.3.2) a v každém průběhu se vnitřní cyklus provede 64krát, aby prošel všechny stavy kodéru (viz sekce 6.2).

Pro výpočet nejpravděpodobnější cesty vedoucí do každého stavu s_i, je třeba znát oba předcházející stavy $s_{(i-1)_1}$ a $s_{(i-1)_2}$, ze kterých se kodér mohl dostat do aktuálního stavu s_i. Dále je třeba určit hodnotu vstupního bitu ($bit = 0..1$), který musel být zakódován, aby po jeho vsunutí na první paměťovou pozici se kodér dostal do aktuálního stavu s_i. Nakonec musíme znát hodnoty dvojice symbolů (out_1 a out_2),

které jsou validním výstupem kodéru pro každý ze dvou momentálně uvažovaných přechodů do aktuálního stavu, aby bylo možné spočítat Hammingovu vzdálenost daného přechodu od přijaté dvojice symbolů. Všechny tyto hodnoty se dají snadno vysledovat v tabulce 6.3. Při realizaci dekodéru tedy musíme mít uloženou obdobnou tabulku v paměti a pracovat s ní, nebo potřebné hodnoty počítat. V uvedené tabulce lze vidět zřejmé pravidelnosti potřebných hodnot. Například, že do stavu 0 až 31 se může kodér dostat pouze pokud zakódoval hodnotu $bit = 0$ a do stavu 31 až 63 pouze pokud zakódoval hodnotu $bit = 1$. Další potřebné závisloti vyjadřují následující vztahy 6.1 až 6.6.

Předcházející stavy stavu aktuálnímu lze určit rovnicemi 6.1 a 6.2. Ovšem pozor, podílem $(s_i/32)$ je myšleno čistě celočíselné dělení, jmenovatel se tedy nesmí zkrátit s hodnotou 32 před závorkou a výsledek podílu se zaokrouhluje dolu. Výsledkem podílu tak může být pouze hodnota 0 nebo 1.

$$s_{(i-1)_1} = 2[s_i - 32\left(\frac{s_i}{32}\right)] \tag{6.1}$$

$$s_{(i-1)_2} = 2[s_i - 32\left(\frac{s_i}{32}\right)] + 1 \tag{6.2}$$

Validní dvojici symbolů představující výstup kodéru pro daný přechod mezi stavy out, rozdělíme na dva jednotlivé symboly sym_1 a sym_2. Tyto symboly spočteme každý samostatně pomocí znalosti výchozího stavu (z rovnice 6.1 resp. 6.2) a znalosti kodéru, resp. polynomů $G1$ a $G2$ popsaných na začátku sekce 6.2. Pro znázornění výpočtu symbolů použijeme binární operace AND ($\wedge$) a negace ($\sim$). Nejprve musíme „spojit“ vstupní bit a pamět kodéru (předešlý stav) do jednoho binárně vyjádřeného čísla Q tak, že na pozici MSB zařadíme vstupní bit a za něj připojíme hodnotu přechozího stavu s_{i-1}:

$$Q = bit \cdot 2^6 + s_{i-1} \tag{6.3}$$

pokud tedy je například $bit = 1$ a $s_{i-1} = 35_{10} = 100011_2$ pak $Q = 1100011_2$. Následně již vypočteme jednotlivé symboly poměrně snadno:

$$sym_1 = (G1 \wedge Q)\ mod\ (2) \tag{6.4}$$

$$sym_2 = \sim [(G2 \wedge Q)\ mod\ (2)] \tag{6.5}$$

$$out = sym_1 \cdot 2^1 + sym_2 \tag{6.6}$$

Pokud máme spočtené validní symboly pro přechod z předchozího stavu s_{i-1} do aktuálního s_i, určíme Hammigovu vzdálenost této dvojice od přijaté dvojice symbolů a tuto hodnotu HD přičteme k HD celé cesty, která vedla až do předcházejícího stavu s_{i-1}. Stejný výpočet provedeme pro druhou možnou cestu vedoucí do aktuálního stavu s_i. Nakonec porovnáme Hammingovy vzdálenosti obou těchto cest a vybereme tu s nižším součtem HD. Méně pravděpodobnou cestu vyloučíme a již ji dále neuvažujeme. Z tabulky 6.3 je také patrné, že validní symboly jednoho přechodu do aktuálního stavu jsou přesně opačné k symbolům přechodu z druhého stavu (00-11, 01-10...). Není tak třeba provádět výpočet symbolů a HD pro oba přechody, ale pouze pro jeden a druhý přechod bude mít HD „opačnou“ (0-2, 2-0, 1-1).

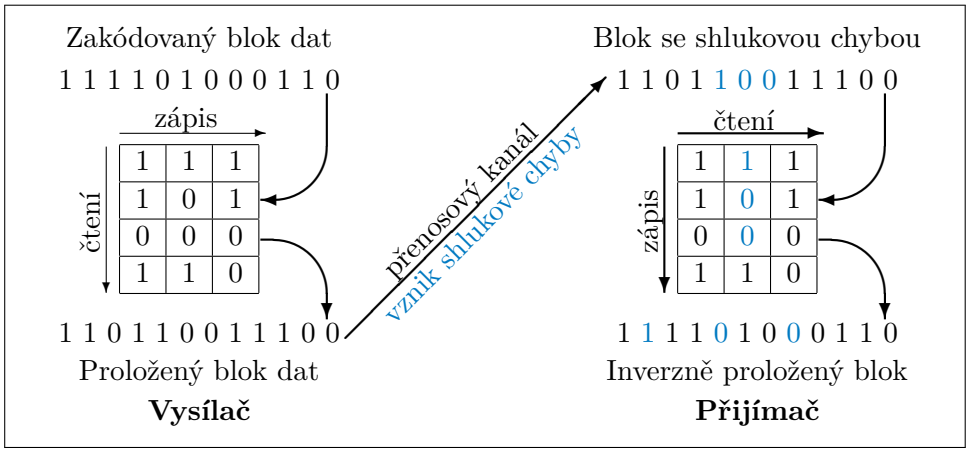
Algoritmus dekódování je implementován v programu funkcí *„ViterbiDecoding“*, do níž vstupuje pouze pole obsahující zakódovaný datový blok zkopírovaný z *bufferu*, výstupem je pole obsahující dekódovaný blok. Dekódování navigační stránky F/NAV probíhá naprosto stejně, pouze se prochází delší mřížka, takže vnější cyklus procházející přes přijaté symboly proběhne 244krát. Celý algoritmus je podrobněji znázorněn na následujícím pseudokódu.

```
ViterbiDecoding(prijato[240])
cesta[0..63].data=0    //Pro ukládání dekódovaných bitů cesty
cesta[0].HD=0    //Počáteční hodnota HD výchozího nultého stavu
cesta[1..63].HD=255    //Počáteční hodnota HD ostatních stavů
for (t=0 ; t<240 ; t=t+2) do
    temp[0..63]=cesta[0..63]    //Vytvoření dočasné pomocné kopie všech cest
    for (s_i=0 ; s_i<64 ; s_i++) do
        if (s_i>31) then
            bit=1
        else
            bit=0
        end
        spočti s_(i-1)_1    //Výpočet podle rovnice 6.1
        spočti s_(i-1)_2    //Výpočet podle rovnice 6.2
        HD1=0    //Hammingova vzdálenost 1. přechodu
        HD2=0    //Hammingova vzdálenost 2. přechodu
        if (prijato[t]!=sym_1) then
            HD1++    //Spočtený symbol se liší, je třeba zvýšit HD
        end
        if (prijato[t+1]!=sym_2) then
            HD1++
        end
        if (HD1==0) then
            HD2=1    //Přiřazení HD druhému přechodu
        else
            if (HD1==2) then
                HD2=0
            else
                HD2=1
            end
        end
          //Zvolení pravděpodobnější cesty
        if (pomoc[ s_(i-1)_1].HD+HD1 >= pomoc[ s_(i-1)_2].HD+HD2) then
            cesta[s_i]=pomoc[ s_(i-1)_2]  //Přiřazení dat a aktualizované HD
        else
            cesta[s_i]=pomoc[ s_(i-1)_1]
        end
        cesta[s_i].data=(cesta[s_i].data«)+bit //Přidání nového bitu
    end
end
return cesta[0].data  //Kodér musí končit ve stavu 0, vrací se tak tato cesta
```

6.3 Prokládání

Prokládání (interleaving) je metoda sloužící k rozprostření shlukové chyby na několik chyb osamocených, které se dají snáze opravit vhodným kódováním. Využívá se matice o definovaných rozměrech, do které se v jednom směru zapisuje posloupnost bitů a v druhém směru se tato posloupnost opět přečte. Tento postup se provede jak na vysílací straně, čímž blok bitového toku poskládáme do jiného pořadí, tak následně na přijímací straně inverzním procesem data opět seřadíme do pořadí původního. Skupina bitů postižených chybou se tak rovnoměrně rozloží po celém bloku na chyby náhodné. Využívají se dva typy prokládání a to blokové (k zápisu a čtení slouží řádky a sloupce matice) a konvoluční (využití diagonál a sloupců). Prokladač (interleaver) se ve vysílači uplatní mezi kódováním a modulací, respektive v přijímači inverzní prokladač (deinterleaver) mezi demodulací a dekodérem.[12][14][15]

Systém Galileo používá blokové prokládání, při kterém se na vysílací straně blok datového toku zapíše do matice po řádcích a přečte po sloupcích, na přijímací straně se zapíše po sloupcích a přečte po řádcích. V případě signálu s navigační zprávou F/NAV se prokládá blok o velikosti 488 bitů do matice o rozměrech 61x8 (řádky x sloupce). U zprávy I/NAV pak blok 240 bitů do matice 30x8.[1][3]



Obrázek 6.5: Příklad prokládání pro matici 4x3

Následující pseudokód ilustruje implementaci algoritmu inverzního prokládání v programu ve funkci *„Deinterleaving“*, jejímž jediným vstupem je datové pole obsahující blok dat, který je třeba inverzně proložit a následně se do něj uloží data ve správném pořadí.

```
Deinterleaving(data[])
matice[8][30];
k=0;    //k = pořadí symbolu v bloku dat
for ( sloupec=0 ; sloupec<8 ; sloupec++ ) do
    for ( řádek=0 ; řádek<30 ; řádek++ ; k++ ) do
        matice[sloupec][řádek]=data[k] //Zápis symbolů do matice po sloupcích
    end
end
k=0;
for ( řádek=0 ; řádek<30 ; řádek++ ) do
    for ( sloupec=0 ; sloupec<8 ; sloupec++ ) do
        data[k]=matice[sloupec][řádek]; //Přečtení z matice po řádcích
        k++;
    end
end
```

6.4 Kontrolní součet

Kontrolní součet je počítán z bloku dat metodou CRC (Cyclic Redundancy Check) a umožňuje nám detekovat chyby vzniklé při přenosu. Součet se spočte na vysílací straně, připojí se za chráněný blok dat, vše se zakóduje a společně přenese na stranu přijímací. Po dekódování se opět nezávisle určí kontrolní součet v přijímači a pokud je totožný s přenášenou hodnotou, tak data byla přenesena bez chyby. Pokud ovšem totožný není, v datech se vyskytuje minimálně jedna chyba. Redundantní součet je u Galilea počítán z užitečných informací, nezahrnuje tedy synchronizační slovo, tail ani rezervované bity, které zatím nemají své využití a nacházejí se v bloku dat až za součtem.[1][3]

Kontrolní součet tvoří 24 paritních bitů (bitově vyjádřený polynom $R(X)$), které jsou zbytkem po binárním algebraickém dělení modulo 2 (operace XOR) polynomu $m(X) * X^{24}$ polynomem $G(X)$. Generující polynom $G(X)$ je násobkem primitivního (dále nerozdělitelného) polynomu $P(X)$.

$$P(X) = X^{23} + X^{17} + X^{13} + X^{12} + X^{11} + X^9 + X^8 + X^7 + X^5 + X^3 + 1 \tag{6.7}$$

$$G(X) = (1 + X) * P(X) \tag{6.8}$$

$$\begin{aligned} G(X)\ mod\ 2 = & X^{24} + X^{23} + X^{18} + X^{17} + X^{14} + X^{11} + X^{10} + \\ & + X^7 + X^6 + X^5 + X^4 + X^3 + X + 1 \end{aligned} \tag{6.9}$$

Polynom $G(X)$ si vyjádříme v binární podobě:

$$G(X)_2 = 1100001100100110011111011 \tag{6.10}$$

Polynom $m(X)$ vyjadřuje posloupnost chráněných bitů, ze kterých je počítán kontrolní součet. Na konec této posloupnosti se za pozici LSB přidá dalších 24 bitů, obsahujících právě spočtený kontrolní součet. Počet chráněných bitů se liší podle typu navigační zprávy. Pro I/NAV se CRC počítá ze 196 bitů každé stránky, pro F/NAV z 214 bitů.[1]

Na následujícím obrázku je zobrazen příklad výpočtu kontrolního součtu z přijaté stránky navigační zprávy v přijímači. Na konci řádku s blokem dat je přenášená hodnota CRC, pokud se námi spočtená hodnota CRC bude rovnat hodnotě přenášené, těchto 24 bitů se postupně v rámci výpočtu vynuluje. Jestliže by se obě hodnoty nerovnaly, bity by se nevynulovaly.

	Přenášená hodnota CRC
100101010101010101010101010101...001	*111001101011100111011001*
⊕110000110010011001111101 1	
010101100111001100101000110101010...001	111001101011100111011001
⊕ 11000011001001100111110 11	
001101111110000000010110000101010...001	111001101011100111011001
⊕ 11000011001001100111110 11	
000001110010100110001001011101010...001	111001101011100111011001
⊕ 11000011001001100111110 11	
000000010011000010111010100110010...001	111001101011100111011001
⋮ ⋮ ⋮	
000000000000000000000000000000000...100	100100111110011011100001
⊕ 110	000110010011001111101 1
000000000000000000000000000000000...010	100010101101010100001101
⊕ 11	000011001001100111110 11
000000000000000000000000000000000...001	100001100100110011111011
⊕ 1	100001100100110011111011
000000000000000000000000000000000...000	*000000000000000000000000*
	Spočtené CRC - přenášené

Obrázek 6.6: Ukázka výpočtu CRC z dat přijatých experimentálním přijímačem

Hodnota CRC se vynulovala, což indikuje, že data po dekódování neobsahují chybu. Za určitých okolností by se mohlo stát, že se hodnota CRC vynuluje i pokud by v datech chyba byla, ale při takto dlouhém kontrolním součtu je to málo pravděpodobné. Algoritmus pro I/NAV je znázorněn na následujícím kódu.

```
CRC(Page[])
m_x[220]=Page[0..113, 120..225]; //Zkopírování potřebných bitů dané stránky
Checksum=0;
for ( i=0 ; i<196 ; i++) do
    if (m_x[i]==0) then
        continue; //Nepočítat pokud je na MSB nulový bit
    end
    for ( j=0 ; j<25 ; j++) do
        m_x[i+j]=m_x[i+j]⊕G_x[j]; //Dělení polynomů modulo 2
    end
end
Checksum=m_x[196..219] //Na posledních 24 bitech je uložena hodnota CRC
return Checksum; //Vrácení spočteného CRC vzhledem k přenášené hodnotě
```

Kapitola 7

Výpočtové algoritmy

Před samotným určením polohy přijímacího zařízení je třeba spočíst několik nezbytných parametrů, typicky jde o zjištění času vyslání signálu z přenášených hodnot a opravu chyb systémových hodin družice. Z výše uvedeného principu dálkoměrné metody (sekce 2.1) víme, že potřebujeme znát přesnou dobu šíření signálu od vysílače k přijímači, abychom byli schopni dopočítat přímou vzdálenost mezi oběma zařízeními. Ovšem v určitých částech přenosového řetězce se signál nešíří přímou cestou, a tak narůstá doba šíření, aniž by se zvyšovala i vzdálenost zařízení. Těmito částmi jsou typicky vysokofrekvenční výstupní obvody družice a různé části zemské atmosféry. Všechna tato zpoždění je třeba korigovat.

7.1 Systémový čas Galileo

Aktuální čas je v navigační zprávě reprezentován dvěmi hodnotami: Time of week (TOW) a Week number (WN). TOW je celočíselná hodnota o délce 20 bitů, obsahující počet sekund od začátku týdne (od půlnoci ze soboty na neděli). Nabývá tedy hodnot od 0 do 604799. Hodnota parametru TOW je vždy vztažena k vzestupné hraně prvního bitu dané stránky navigační zprávy, ve které je parametr přenášen. WN je také celočíselná hodnota o délce 12 bitů, přenášející informaci o počtu týdnů od začátku počítání GST, tedy od půlnoci ze soboty 21. srpna na neděli 22. srpna 1999, kdy začal 1. týden GST. Nabývá hodnot od 0 do 4096, po dosažení maximální hodnoty je WN vynulován.[1]

V programu jsou vytvořeny globální struktury pro ukládání hodnot času pro každou družici. Při příjmu WT, který přenáší parametr TOW, se tato hodnota uloží do proměnné určené pro danou družici a protože je TOW vztažena k začátku stránky a my stránku zpracováváme až po jejím kompletním příjmu, musíme k hodnotě TOW přičíst čas, který uběhl během vysílání zbytku stránky a dalšího synchronizačního slova, tedy $260 \cdot 4ms = 1,04s$. Hodiny každé družice se inkrementují po příjmu každého bitu o $4ms$. Stejně tak je vytvořena proměnná reprezentující hodiny přijímače, jež se po příjmu první hodnoty TOW synchronizuje s danou družicí a dále se inkrementuje samostatně.

Časový krok hodin $4ms$ je pro družicovou navigaci samozřejmě příliš velký, jednoduchým výpočtem zjistíme, že za $4ms$ signál urazí vzdálenost téměř 1200km. Proto pokud chceme spočítat polohu uživatele (viz sekce 7.7), musíme v okamžiku zavolání funkce pro tento výpočet určit čas družice naprosto přesně. K tomu jsou využity časové korekce a dálkoměrný PRN kód, jehož perioda je stejně dlouhá, jako doba

vysílání jednoho bitu. O zpracování PRN kódu, stejně jako o korelaci a demodulaci se starají jiné části přijímače. Ty ukládají do globální struktury informace o počtu chipů, které uběhly od přidání posledního bitu do *bufferu* a zlomkovou část aktuálního a zatím necelého chipu kódu. Pro signál E1-B je PRN kód dlouhý 4092 chipů, s periodou právě $4ms$, takže jsme schopni výpočtem určit čas s přesností v řádu nanosekund.

Přepočet GST-UTC

Pomocí navigační zprávy jsou také přenášeny parametry nezbytné pro případný přepočet času GST na koordinovaný světový čas UTC. GST narozdíl od UTC nepoužívá přestupné sekundy, takže vzájemná diference mezi těmito časovými soustavami není fixní a v průběhu času se zvyšuje.

Přepočet GST-GPSt

Přepočet mezi časovými základnami družicových systémů Galileo a GPS je o poznání snažší než přepočet na UTC, protože obě tyto soustavy neuvažují přestupné sekundy, podobně jako TAI (mezinárodní atomový čas). Jejich vzájemná diference je tak až na malou proměnlivou odchylku konstantní. Potřebné parametry jsou také přenášeny pomocí navigační zprávy.

7.2 Časové korekce

Časová základna každé družice se nepatrně liší od referenčního času celého systému. Odchylku této základny počítá řídící segment a parametry pro korekci následně vysílá daná družice.[9] Odchylku časové základny získáme výpočtem polynomu druhého řádu:

$$\Delta t_{sv} = a_{f0} + a_{f1}(t - t_{0c}) + a_{f2}(t - t_{0c})^2 + \Delta t_r \tag{7.1}$$

kde t_{0c} je referenční čas korekčních parametrů, a_{f0}, a_{f1} a a_{f2} jsou korekční parametry, přenášené v rámci navigační zprávy (I/NAV v rámci WT 4), t je GST dané družice a Δt_r je relativistická korekce.[1]

Relativistické korekce vycházejí z Einsteinovy obecné a speciální teorie relativity. Ze speciální teorie relativity víme, že čas na objektu (družici), který se oproti nám pohybuje konstantní rychlostí, plyne pomaleji než náš (na Zemi). Obecná teorie zase říká, že hodiny na družici běží rychleji než na povrchu Země, protože na oběžné dráze družice je menší gravitační potenciál než na Zemi. Důsledkem těchto jevů je nepatrně rychlejší plynutí času na družici než na povrchu Země. Částečné kompenzace se dosahuje snížením kmitočtu hodin družice. To by byla dostačující kompenzace v případě přesně kruhové oběžné dráhy, ovšem orbita satelitu je eliptická, a tak po jejím obvodu není gravitační potenciál konstantní.[4] Z tohoto důvodu je třeba počítat relativistický korekční člen časové základny Δt_r:

$$\Delta t_r = F \cdot e \cdot \sqrt{A} \cdot sin(E) \tag{7.2}$$

kde e je excentricita, A délka hlavní poloosy dráhy družice, E excentrická anomálie a konstanta $F = -2\sqrt{\mu}/c^2 = -4,442807309 \cdot 10^{-10} s/m^{1/2}$.[1]

O počítání výše uvedených časových korekcí se stará funkce *„Time_Correction“*, jejíž vstupní parametry jsou SVID a E. Výstupem je hodnota časové korekce.

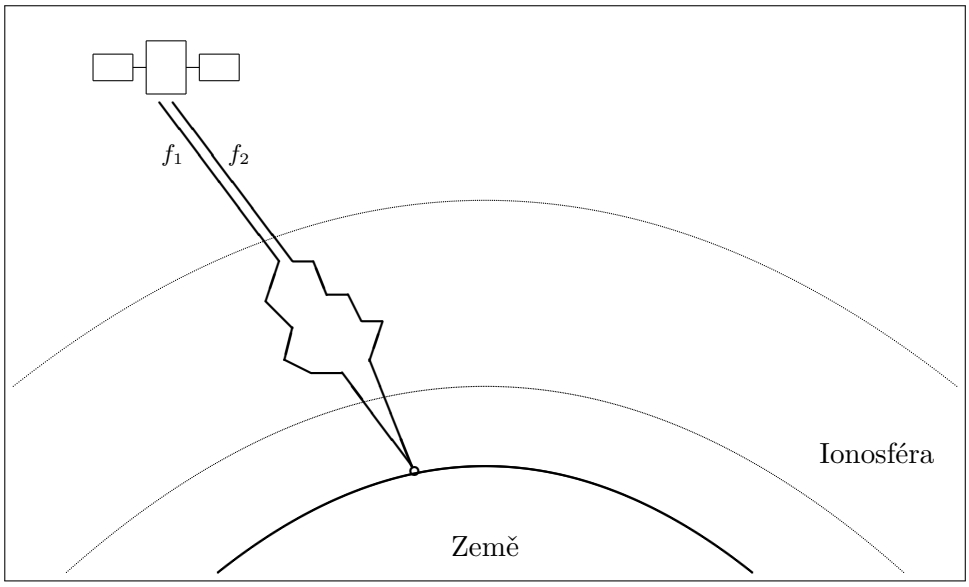
7.3 Skupinové zpoždění

Určité zpoždění signálu vzniká na výstupních obvodech samotné družice (Broadcast group delay (BGD)), jde zejména o kabely a jiná vedení, filtry nebo anténu. Doba průchodu přes tyto obvody je navíc rozdílná pro různé frekvence signálu. Eliminovat toto zpoždění šíření signálu se dá bez výpočtu za předpokladu dvoufrekvenčního příjmu signálu, ovšem pokud přijímáme signál jen na jedné frekvenci, musíme zpoždění spočítat z poskytovaných parametrů navigační zprávy.[16]

Parametr BGD je definován pro dvojici signálů f_1 a f_2 ($BGD(f_1, f_2)$), tedy pro $E1$ a $E5a$ ($BGD(E1, E5a)$) nebo pro $E1$ a $E5b$ ($BGD(E1, E5b)$). Při příjmu na frekvenci f_1 se od korekčního členu Δt_{sv} pouze odečte parametr $BGD(f_1, f_2)$. V případě příjmu na frekvenci f_2 je třeba upravit korekci Δt_{sv} odečtením členu $((\frac{f_1}{f_2})^2 BGD(f_1, f_2))$.[1] Skupinové zpoždění se uplatňuje také v rámci funkce *„Time_Correction“*.

7.4 Ionosférické korekce

Ionosféra je vrstva atmosféry nalézající se v rozmezí přibližně od 50 do 1000km nad povrchem Země. Sluneční paprsky v této vrstvě ionizují částice plynu, které uvolňují volné elektrony a ionty. Vlastnosti, jako je míra ionizace nebo tloušťka vrstvy, se mění v závislosti na sluneční aktivitě, a tedy i v závislosti na denní a roční době.[4] Ionizované částice ovlivňují elektromagnetické vlnění a dochází k refrakci signálu.[3] V důsledku refrakce se signál nešíří ionosférou přímou cestou od vysílače k přijímači.



Obrázek 7.1: Ionosférická refrakce

Prodloužení dráhy signálu v důsledku refrakce v ionosféře lze korigovat použitím dvoufrekvenčního příjmu, nebo v případě příjmu pouze na jedné frekvenci pomocí definovaných ionosférických modelů, kterými lze spočítat průměrný obsah elektronů (TEC - total electron content) v závislosti na čase a poloze, a dále určit zpoždění. Například GPS využívá tzv. Klobucharův model, který dokáže redukovat vliv iono-

sféry o 50%.[7] Systém Galileo používá novější a propracovanější model NeQuick G, který počítá s trojrozměrným prostorem a dokáže korigovat vliv ionosféry až o 70%. NeQuick G vychází z modelu NeQuick 2, který je doporučován v ITU-R. Výpočet je prováděn na základě vstupního parametru efektivní ionizační úrovně A_z:

$$A_z = a_{i0} + a_{i1} \cdot MODIP + a_{i2} \cdot (MODIP)^2 \tag{7.3}$$

kde parametry a_{i0}, a_{i1} a a_{i2} jsou přenášeny v rámci navigační zprávy a $MODIP$ je modifikovaný sklon zeměpisné šířky v místě přijímače.[17] Pro družici v zenitu a v odpoledním čase může ionosférické zpoždění dosahovat 5 až 15 metrů.[4] Ovšem pro satelit v méně příznivém úhlu se ještě zvyšuje.

7.5 Troposférické korekce

K dalšímu zpoždění signálu dochází v troposféře, v níž se signál šíří pomaleji než ve volném prostoru.[4] Troposféra je spodní část atmosféry a naneštěstí není disperzní pro signály o frekvencích do 15GHz.[18] Protože refraktivita troposféry není na GNSS kmitočtech frekvenčně závislá, nelze ji kompenzovat pomocí dvoufrekvenčního měření. Vliv troposféry závisí na rozložení dvou hlavních složek - suchých plynů a vodních par. Refrakce se tedy dá modelovat pomocí parametrů jako je teplota, tlak a vlhkost. Přenášet pomocí satelitů nebo měřit tyto atmosférické údaje přijímačem je velmi nepraktické, protože jsou silně závislé na poloze na Zemi a rychle proměnlivé. Proto se používá průměrný meteorologický model standartní atmosféry, jehož data závisí na zeměpisné poloze a dni v roce. Chyba vzdálenosti může v důsledku neuvažování vlivu troposféry dosahovat přibližně 2,5 až 25 metrů.[4]

Implementace troposférických a ionosférických korekčních algoritmů není předmětem této práce, ovšem je třeba je znát a při výpočtu polohy uživatele je brát v úvahu. Protože chyba způsobená neuvažováním těchto zpoždění může dosahovat v součtu až několika desítek metrů ve vzdálenosti vysílače a přijímače.

7.6 Určení polohy družice

Polohu družice v určitém čase lze spočítat dle níže uvedeného postupu, definovaného v ICD [1], z přenášených efemerid. Pro získání skutečné hodnoty je třeba u všech parametrů uvažovat vynásobení celočíselné přenášené hodnoty definovaným činitelem (viz sekce 5.2). Dále je třeba čtveřice následujících konstant:

Geocentrická gravitační konstanta Země:	$\mu = 3,986004418 \cdot 10^{14} \frac{m^3}{s^2}$
Střední rychlost rotace Země:	$\omega_E = 7,2921151467 \cdot 10^{-5} \frac{rad}{s}$
Rychlost světla:	$c = 299792458 \frac{m}{s}$
Poměr obvodu kruhu k jeho průměru:	$\pi = 3,1415926535898$

Délku hlavní poloosy oběžné dráhy určíme snadno z přenášeného parametru $A^{1/2}$, který je třeba vynásobit činitel 2^{-19}:

$$A = (A^{1/2})^2 \tag{7.4}$$

Následně spočteme střední pohyb n_0:

$$n_0 = \sqrt{\frac{\mu}{A^3}} \tag{7.5}$$

Určíme čas od vztažného času efemerid podle následující rovnice 7.6, kde t_{0e} je přenášená hodnota referenčního času efemerid, pro získání skutečné hodnoty je třeba ji vynásobit číslem 60 a t je systémový čas družice, ve kterém chceme znát její polohu:

$$t_k = t - t_{0e} \tag{7.6}$$

Korigovaný střední pohyb n spočítáme pomocí n_0 a přenášeného parametru Δn, což je rozdíl středního pobyhu od vypočítané hodnoty. Δ_n je třeba ještě násobit činitelem $2^{-43} \cdot \pi$, protože jednotka tohoto parametru je $(\frac{semi-circles}{s})$, což není odvozená jednotka SI a lze ji přepočíst na radiány jako $1(semi - circle) = \pi(rad)$.

$$n = n_0 + \Delta n \tag{7.7}$$

Poté lze vypočíst střední anomálii M ze střední anomálie v referenčním čase M_0. Parametr M_0 je třeba opět násobit, tentokrát faktorem $2^{-32} \cdot \pi$. A protože se parametry n_0 a n v dalších rovnicích nevyužijí, lze jejich výpočet zahrnout přímo do následující rovnice 7.8.

$$M = M_0 + n \cdot t_k = M_0 + \left(\sqrt{\frac{\mu}{A^3}} + \Delta n\right) \cdot t_k \tag{7.8}$$

Pro získání excentrické anomálie E musíme spočítat Keplerovu rovnici 7.9, kde e je přenášená hodnota excentricity, násobená činitelen 2^{-33}.

$$M = E - e \cdot sin(E) \tag{7.9}$$

Výše uvedenou Keplerovu rovnici 7.9 je třeba řešit iterativně, tedy několikrát dokola provádět stejný výpočet a v každé iteraci i tak zpřesňovat výsledek, dokud nedosáhneme požadované tolerance přesnosti $\xi = |E_{i-1} - E_i|$. V první iteraci položíme například $E_1 = M$ a $E_{i-1} = E_i$, dále počítáme následující rovnici 7.10 v i krocích, dokud nedosáhneme ξ nebo maximálního počtu iterací. V případě výše uvedených počátečních podmínek ($E_1 = M$) rovnice konverguje velmi rychle, průměrně ve třech až čtyřech cyklech.

$$E_i = M + e \cdot sin(E_{i-1}) \tag{7.10}$$

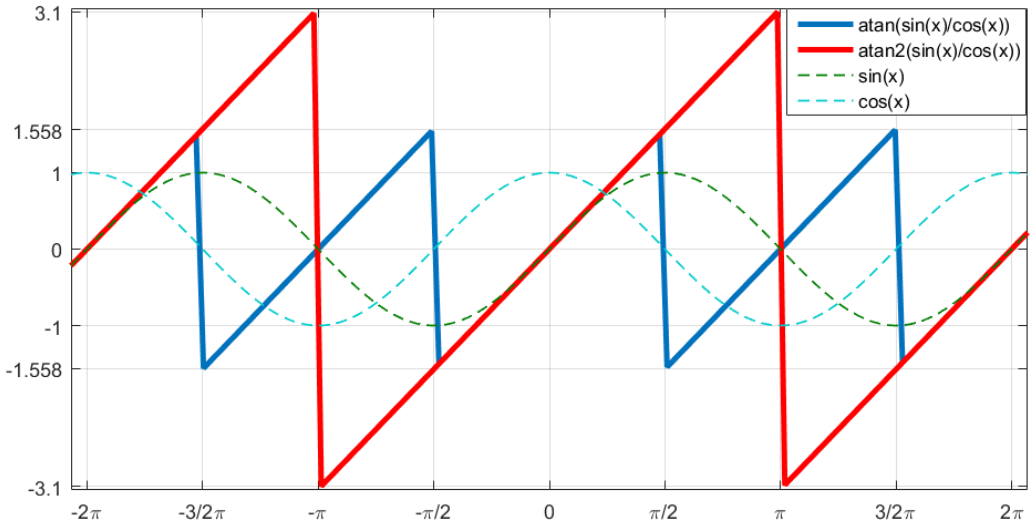
Iterační řešení tohoto výpočtu je znázorněno na následujícím kódu:

```
E=M;
while (|prevE-E| > ξ)) do
    //ξ např. 10^-9
    prevE=E;
    E=M+e*2^-33*sin(prevE);
    kep_iter++;
    if (kep_iter>4) then
        break;
    end
end
```

Následně vypočteme pravou anomálii ν:

$$\nu = tan^{-1}\left\{\frac{sin\ \nu}{cos\ \nu}\right\} = tan^{-1}\left\{\frac{\frac{(\sqrt{1-e^2}\cdot sin(E))}{(1-e\cdot cos(E))}}{\frac{(cos(E)-e)}{(1-e\cdot cos(E))}}\right\} \tag{7.11}$$

Řešení této rovnice 7.11 není tak triviální, jak by se na první pohled mohlo zdát. Z důvodu podílu periodických funkcí v argumentu funkce arkus tangens nelze v programovacím jazyku C (ani dalších) využít běžnou matematickou funkci *atan* (resp. *arctan*), ale funkci *atan*2 (resp. *arctan*2), jenž respektuje znaménko dělence i dělitele a ne pouze znaménko výsledku jejich podílu. Rozdíl ve výsledcích těchto dvou funkcí je vidět na následujícím obrázku 7.2.



Obrázek 7.2: Srovnání průběhu funkcí *„atan“* a *„atan2“*

Součtem pravé anomálie ν a argumentu perigea ω, jenž vynásobíme faktorem $2^{-31}\cdot\pi$, získáme argument šířky Φ:

$$\Phi = \nu + \omega \tag{7.12}$$

Následujícími třemi rovnicemi 7.13, 7.14 a 7.15 dostaneme korekci argumentu šířky δu, korekci poloměru δr a korekci inklinace δi:

$$\delta u = C_{us}\cdot sin(2\Phi) + C_{uc}\cdot cos(2\Phi) \tag{7.13}$$

$$\delta r = C_{rs}\cdot sin(2\Phi) + C_{rc}\cdot cos(2\Phi) \tag{7.14}$$

$$\delta i = C_{is}\cdot sin(2\Phi) + C_{ic}\cdot cos(2\Phi) \tag{7.15}$$

Kde C_{us} a C_{uc} jsou amplitudy sinové a kosinové části harmonické korekce argumentu šířky, C_{is} a C_{ic} jsou amplitudy sinové a kosinové části harmonické korekce inklinačního úhlu. Tyto čtyři parametry se násobí faktorem 2^{-29}. C_{rs} a C_{rc} jsou amplitudy sinové a kosinové část harmonické korekce poloměru oběžné dráhy, násobené faktorem 2^{-5}.

Dále zkorigujeme argument šířky Φ o korekci δu:

$$u = \Phi + \delta u \tag{7.16}$$

Korigovaný poloměr oběžné dráhy r:

$$r = A(1 - e \cdot cos(E)) + \delta r \tag{7.17}$$

Korigovanou inklinaci i spočteme následujícím výrazem 7.18, kde $i_0 \cdot 2^{-31} \cdot \pi$ je inklinační úhel v referenčním čase a $\dot{i}$ je rychlost změny inklinačního úhlu s měřítkem $\pi \cdot 2^{-43}$.

$$i = i_0 + \delta i + (\dot{i})t_k \tag{7.18}$$

Souřadnice x, y v orbitální rovině družice dostaneme z následují dvojice rovnic 7.19:

$$\begin{aligned} x' &= r \cdot cos(u) \\ y' &= r \cdot sin(u) \end{aligned} \tag{7.19}$$

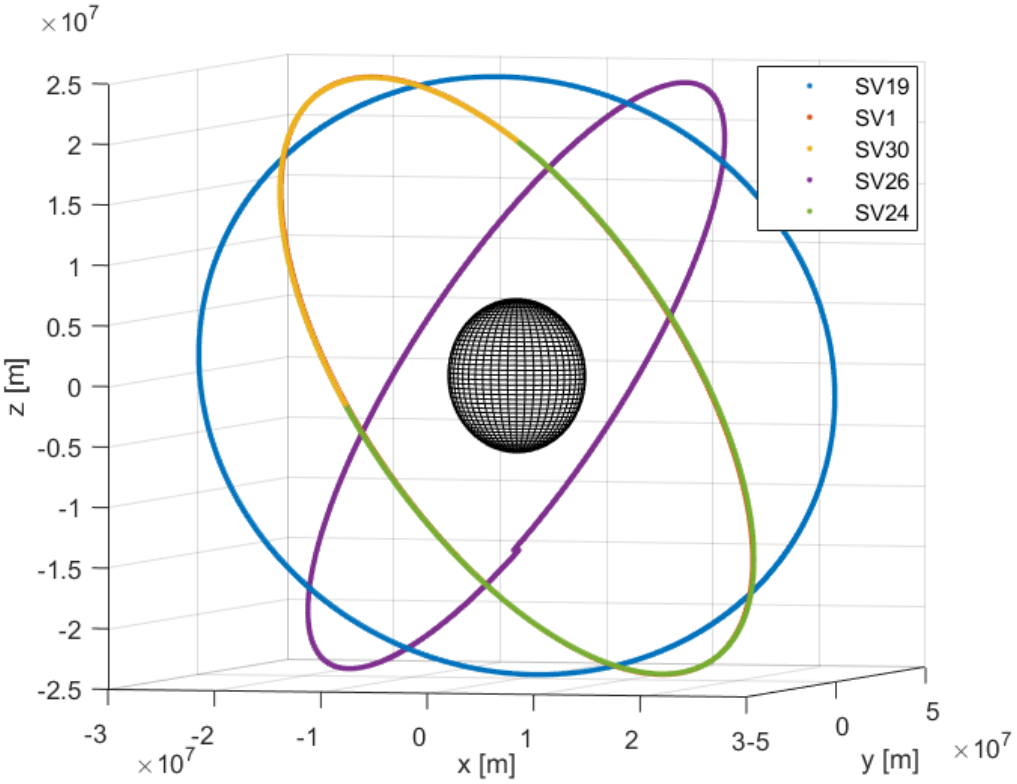
Následně spočteme korigovanou délku vzestupného uzlu Ω, kde Ω_0 s faktorem $\pi \cdot 2^{-31}$ je zeměpisná délka vzestupného uzlu orbitální roviny v referenčním čase a $\dot{\Omega} \cdot \pi \cdot 2^{-43}$ je rychlost změny rektascenze, což je úhel mezi deklinační rovinou jarního bodu a deklinační rovinou družice:

$$\Omega = \Omega_0 + (\dot{\Omega} - \omega_E)t_k - \omega_E \cdot t_{0e} \tag{7.20}$$

Z poslední trojice vztahů dostaneme požadovanou polohu družice x, y, z v metrech v souřadném systému ECEF:

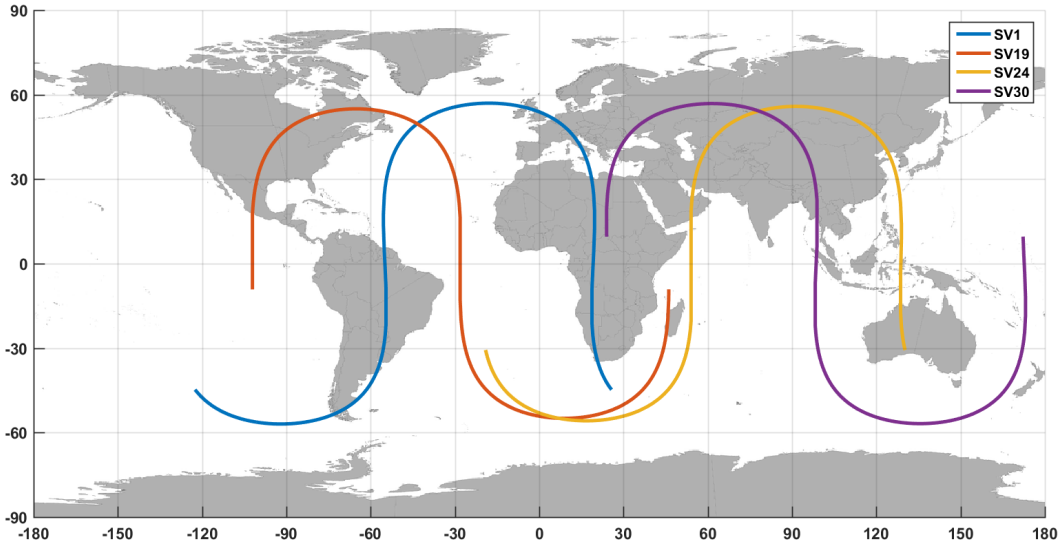
$$\begin{aligned} x &= x' \cdot cos(\Omega) - y' \cdot cos(i) \cdot sin(\Omega) \\ y &= x' \cdot sin(\Omega) + y' \cdot cos(i) \cdot cos(\Omega) \\ z &= y' \cdot sin(i) \end{aligned} \tag{7.21}$$

Následující obrázek 7.3 zobrazuje přibližné dráhy některých družic, které byly vypočteny vytvořeným algoritmem na základě efemerid, přenášených v navigační zprávě a v jejím almanachu, ovšem rychlost rotace Země byla nastavena na nulovou hodnotu, aby celá soustava nerotovala. Tato zpráva byla získána příjmem reálného signálu družice a jeho pozdějším zpracováním (dekódováním, extrakcí dat...) pomocí vytvořených funkcí. Funkce pro výpočet polohy byly volány v cyklu, v němž se pokaždé inkrementoval čas o 1 vteřinu, čas se takto postupně navýšil přibližně o 14 hodin, což odpovídá době oběhu družice kolem Země. Vykreslené polohy jsou přibližné, protože efemeridy mají danou platnost pouze pár hodin a zde byly použity i mimo určenou platnost. Dráhy družic číslo 1, 24 a 30 se překrývají, protože obíhají po stejné oběžné dráze.



Obrázek 7.3: Vypočtené oběžné dráhy vybraných družic

Obdobným způsobem jako na obrázku 7.3 byly vypočítány i dráhy družic vykreslené na obrázku 7.4, ovšem tentokrát byla uvažována rychlost rotace Země a vypočtené souřadnice x, y, z byly převedeny na zeměpisnou šířku a délku a následně vyneseny do zeměpisné sítě.



Obrázek 7.4: Vypočtené dráhy vybraných družic v zeměpisných souřadnicích

7.6.1 Rotace souřadnic družice

Jak je uvedeno v sekci 4.2, soustava ECEF není inerciální a otáčí se společně s planetou Zemí. Ovšem satelit, jehož polohu počítáme, se neotáčí společně se souřadnou soustavou, navíc v této soustavě není známa konstantní rychlost šíření signálu. Pokud tedy počítáme polohu družice a poté polohu uživatele, je vhodné využít inerciální soustavu. Abychom nemuseli přepočítávat několikrát souřadnice do jiné soustavy, je výhodné zvolit takovou inerciální soustavu, která je v okamžiku výpočtu polohy uživatele t_{rx} (tedy v okamžiku příjmu signálu) totožná se soustavou ECEF a zastavit („zmrazit“) její rotaci. Protože polohu družice počítáme v čase vyslání signálu t_{tx} v soustavě ECEF, tak za dobu šíření signálu $(t_{rx} - t_{tx})$ od vysílače k přijímači se Země společně s celou soustavu ECEF otočí o úhel Θ. O stejný úhel Θ tedy musíme následně otočit i polohu družice kolem osy z:

$$\Theta = \dot{\omega}_E(t_{rx} - t_{tx}) \tag{7.22}$$

Souřadnice družice v inerciální soustavě $(x_{eci}, y_{eci}, z_{eci})$ použité pro následný výpočet polohy uživatele tak získáme z následujícího vztahu 7.23 [3]:

$$\begin{bmatrix} x_{eci} \\ y_{eci} \\ z_{eci} \end{bmatrix} = \begin{bmatrix} cos\Theta & sin\Theta & 0 \\ -sin\Theta & cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{7.23}$$

Výpočet polohy družice v programu probíhá v rámci funkce *„SV_coordinates“*, aplikováním výše uvedených rovnic v daném pořadí. Výpočet polohy družice z efemerid přenášených almanachem je totožný, ovšem parametry které almanach neobsahuje (například korekční členy) se nahrazují nulou, pro výpočet polohy z efemerid je určena funkce *„almanac_SV_coordinates“*. Následné otočení spočtených souřadnic okolo osy z o úhel Θ provádí funkce *„system_rotation“*.

7.7 Návrh výpočtu polohy uživatele

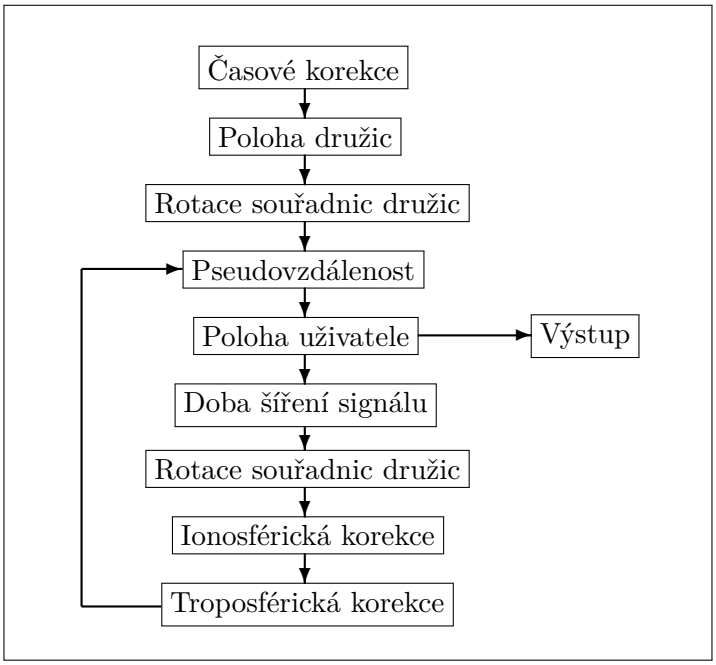
K určení polohy přijímače se využije dálkoměrná metoda (viz sekce 2.1). Příjem signálu tedy musí probíhat od čtyř družic zároveň. K určení pseudovzdáleností a následnému určení polohy se přistoupí až v okamžiku, kdy máme dekódován a uložen dostatečný počet dat (kompletní efemeridy, všechny korekční členy…) z navigačních zpráv všech čtyř družic. Pseudovzdálenosti se určí pomocí generování kopie PRN kódu dané družice a z časového posunu této kopie oproti kódu přijímanému. Protože PRN kód na signálu E1-B má periodu pouze $4ms$, tak z měření získáme posun repliky kódu modulo $4ms$.

Aplikujeme korekce časových základen družic a vypočteme polohy družic z přijatých efemerid v přesném čase vyslání signálu. Dále je třeba polohy družic pootočit okolo osy z kvůli rotaci souřadné soustavy (viz sekce 7.6.1), k tomuto otočení je třeba dosadit přesnou dobu šíření signálu, která ovšem v důsledku nesynchronizace časové základny systému a přijímače není známa. Proto je třeba celý výpočet polohy uživatele provádět iterativně a v první iteraci dosadit do funkce otočení souřadnic družice přibližnou hodnotu šíření signálu (například $85ms$). Takto získané polohy

družic a změřené pseudovzdálenosti dosadíme do rovnice 2.2 a vypočteme první odhad polohy uživatele.

Ze znalosti přibližné polohy uživatele a známých poloh družic určíme přesnější vzdálenost mezi přijímačem a vysílačem, a tedy i přesnější dobu šíření signálu z družice k přijímači. V druhé iteraci tak opět provedeme rotaci souřadnic družic kolem osy z, ovšem již o korigovaný úhel. Protože již máme i odhadnutý údaj o poloze uživatele, můžeme aplikovat také ionosférický a troposférický model, které jsou závislé na poloze a korigovat tak refrakční zpoždění signálu při průchodu zemskou atmosférou.

Stejný postup se provede i v dalších iteracích, dokud není dosaženo požadované přesnosti, resp. minimální odchylky mezi hodnotou získanou v předchozí a aktuální iteraci. Celý návrh popsaného algoritmu je znázorněn na následujícím obrázku 7.5 ve formě vývojového diagramu. Naznačený cyklus probíhá dokud není dosaženo tolerance odchylky mezi předchozí a aktuální hodnotou pseudovzdálenosti.



Obrázek 7.5: Schéma návrhu algoritmu výpočtu polohy uživatele

Kapitola 8

Závěr

Všechny vytvořené algoritmy jsou otestovány a plně funkční. Ověření správnosti výpočtů bylo provedeno několika způsoby.

Funkce pro dekódování a prokládání byly nejprve testovány na ukázkové sekvenci jedné poloviny stránky navigační zprávy I/NAV, uvedené v ICD [1, str. 63] v nezakódované, zakódované i v interleaverem proložené podobě. Po průchodu této proložené sekvence vytvořeným deinterleaverem byl výsledek shodný s neproloženou sekvencí. Následně se výsledek vložil do funkce pro dekódování a výstup byl opět shodný s daty v ICD. Poté byly vytvořeny a obdobně otestovány funkce simulující kodér a interleaver, těmito funkcemi bylo zakódováno a proloženo několik náhodných datových sekvencí a následně opět úspěšně dekódováno.

Další testování bylo prováděno na několika sadách dat, získaných přijetím a demodulováním signálu z generátoru GNSS signálů a příjmem signálů družic z oblohy, pomocí vyvíjeného multikonstelačního GNSS přijímače. Data extrahovaná a uložená, po dekódování, z navigační zprávy se shodovala s parametry nastavenými na GNSS generátoru. Stejně tak se data almanachu, získaná z navigační zprávy družice, shodovala s volně dostupným almanachem na internetových stránkách GSA [19]. Přenášené informace o čase také odpovídaly skutečnému času měření signálu. Funkce aplikující metodu CRC také funguje, protože spočtená hodnota kontrolního součtu byla shodná s hodnotou přenášenou v případě kdy získaná data odpovídala skutečnosti. Naopak pokud se data neshodovala, tak spočtená hodnota kontrolního součtu byla rozdílná a další zpracování dané stránky se ukončilo.

Vypočtené oběžné dráhy družic odpovídají definovaným parametrům drah jak výškou nad povrchem Země, tak rozptylem v zeměpisné šířce, která se pohybuje přibližně mezi 56° jižní šířky a 56° severní šířky. Odpovídá tak inklinaci oběžných drah k rovníku.

Vývoj algoritmů je plánován i nadále, zejména vytvoření dalších nezbytných funkcí a korekčních modelů, potřebných k určení přesné polohy uživatele. Také budou zhodnoceny možnosti další optimalizace a rozšíření stávajících algoritmů, například přesná lokalizace a oprava chybného bitu pomocí CRC, namísto pouhé detekce nebo zobecnění některých funkcí pro snadnější využití i v jiných aplikacích.

Příloha A

Literatura

[1] EUROPEAN UNION. *European GNSS (Galileo) Open Service Signal In Space Interface Control Document*. European Union, 1.3 edition, December 2016. [online] Dostupné z: http://www.gsc-europa.eu/system/files/galileo_documents/Galileo-OS-SIS-ICD.pdf.

[2] HRDINA, Zdeněk, Petr PÁNEK a František VEJRAŽKA. *Rádiové určování polohy (Družicový systém GPS)*. České vysoké učení technické v Praze - vydavatelství ČVUT, Praha, 1995. ISBN 80-01-01386-3.

[3] KAPLAN, Elliott and Christopher HEGARTY. *Understanding GPS: principles and applications*. Artech House, Boston, 2nd edition, 2006. ISBN 1-58053-894-0.

[4] MISRA, Pratap and Per ENGE. *Global positioning system: signals, measurements, and performance*. Ganga-Jamuna Press, Lincoln, 2001. ISBN 0-9709544-0-9.

[5] Department of DEFENSE, Department of HOMELAND SECURITY, and Department of TRANSPORTATION. *Federal radionavigation plan*. National Technical Information Service, Springfield, 2017. [online] Dostupné z: https://www.navcen.uscg.gov/pdf/FederalRadioNavigationPlan2017.pdf.

[6] National Coordination Office for Space-Based Positioning, Navigation, and Timing. *Space Segment*, 2018. [online] Dostupné z: https://www.gps.gov/systems/gps/space/.

[7] HOFMANN-WELLENHOF, Bernhard, Herbert LICHTENEGGER and Elmar WASLE. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer-Verlag, Vienna, c2008. ISBN 978-3-211-73012-6.

[8] ČESKÝ KOSMICKÝ PORTÁL. *Čínský navigační systém Beidou / Compass*. [online] Dostupné z: http://www.czechspaceportal.cz/3-sekce/gnss-systemy/gnss-mimo-evropu/cinsky-beidou---compass/.

[9] KOVÁŘ, Pavel. *Družicová navigace*. České vysoké učení technické v Praze, Česká technika - nakladatelství ČVUT, Praha, 2016. ISBN 978-80-01-05989-0.

[10] EUROPEAN SPACE AGENCY. *Galileo services*, 2010. [online] Dostupné z: http://www.esa.int/Our_Activities/Navigation/Galileo/Galileo_services/.

[11] NAVIPEDIA. *Galileo Navigation Message*, 2017. [online] Dostupné z: http://www.navipedia.net/index.php?title=Galileo_Navigation_Message&oldid=13833.

[12] DOBEŠ, Josef a Václav ŽALUD. *Moderní radiotechnika*. BEN - technická literatura, Praha, 2006. ISBN 80-7300-132-2.

[13] VITERBI, Andrew J. and Jim K. OMURA. *Principles of digital communication and coding*. McGraw-Hill, New York, c1979. ISBN 0-07-067516-3.

[14] DUMAS,Jean-Guillaume, Jean-Louis ROCH, Éric TANNIER and Sébastien VARRETTE. *Foundations of Coding: Compression, Encryption, Error Correction*. John Wiley & Sons, Inc., New Jersey, c2015. ISBN 978-1-118-88144-6.

[15] HOUGHTON, A. *Error Coding for Engineers*. Springer Science+Business Media, New York, c2001. ISBN 978-1-4613-5589-2.

[16] NAVIPEDIA. *Instrumental Delay*, 2011. [online] Dostupné z: http://www.navipedia.net/index.php?title=Instrumental_Delay&oldid=11608.

[17] EUROPEAN COMMISSION. *European GNSS (Galileo) Open Service-Ionospheric Correction Algorithm for Galileo Single Frequency Users*. European Commission, 1.2 edition, September 2016. [online] Dostupné z: https://www.gsc-europa.eu/system/files/galileo_documents/Galileo_Ionospheric_Model.pdf.

[18] HOPFIELD H.S. Two-quartic tropospheric refractivity profile for correcting satellite data. *Journal of Geophysical Research*, vol. 74(18), 1969.

[19] European GNSS Agency. *Almanac data*. [online] Dostupné z: https://www.gsc-europa.eu/system-status/almanac-data.

Příloha B

Zdrojové kódy

B.1 Hlavičkový soubor

: process_GALILEO_data.h

```
1  /** Author: Tomas Vankat */
2  #ifndef PROCESS_GALILEO_DATA_H_
3  #define PROCESS_GALILEO_DATA_H_
4
5  int process_GALILEO_data(uint64_t[],uint8_t);
6  void Deinterleaving(uint8_t[]);
7  void ViterbiDecoding(uint8_t[],uint8_t[]);
8  int Dekodovani(uint64_t[],uint8_t[]);
9  void sortWT0(uint8_t[],uint8_t);
10 void sortWT1(uint8_t[],uint8_t);
11 void sortWT2(uint8_t[],uint8_t);
12 void sortWT3(uint8_t[],uint8_t);
13 void sortWT4(uint8_t[],uint8_t);
14 void sortWT5(uint8_t[],uint8_t);
15 void sortWT6(uint8_t[],uint8_t);
16 void sortWT7(uint8_t[],uint8_t);
17 void sortWT8(uint8_t[],uint8_t);
18 void sortWT9(uint8_t[],uint8_t);
19 void sortWT10(uint8_t[],uint8_t);
20 int CRC(uint8_t []);
21 double Time_Correction(uint8_t,double);
22 double SV_coordinates(uint8_t);
23 double GST_UTC(uint8_t);
24 int readvalue(uint8_t, uint8_t, uint8_t [], uint8_t);
25 void system_rotation(uint8_t, double);
26 void almanac_SV_coordinates(uint8_t,uint8_t,int16_t,uint16_t,
       int16_t,int16_t,int16_t,int16_t,int16_t,uint16_t,uint8_t);
27 void store_almanac_data(uint8_t,uint8_t);
28 double User_pos();
29
30 const double PI=3.1415926535898;
31 const double my=398600441800000; //geocentricka gravitacni konst.
32 const double omega_e=0.000072921151467; //str. rychl. rotace Zeme
33 const double c=299792458; //rychlost svetla
34
35 uint8_t initialization=0;
36 double Eccentr_anomal;
```

```
37 double receiver_time;
38 double time0[36]; //doba sireni signalu od druzic
39
40 struct {
41     double daytime; //cas dne v sekundach
42     uint8_t day; //0..Nedele - 6..Sobota
43     uint8_t date;
44     uint8_t month;
45     uint16_t year;
46 }UTC;
47
48 struct {
49     int week; //cislo tydne epochy GST
50     double time; //cas druzice v sekundach
51     int64_t precise_time; //cas druzice v pikosekundach
52 }GST[36];
53
54 struct SVcoordin{ //souradnice druzic
55     double X;
56     double Y;
57     double Z;
58 }SV_coor[36];
59
60 struct alma{       //efemeridy z almanachu
61     uint8_t IODa;
62     uint8_t WNa;
63     uint16_t t_0a;
64     uint8_t SVID;
65     int16_t delta_sqrtA;
66     uint16_t e;
67     int16_t omega;
68     int16_t d_i;
69     int16_t omega_0;
70     int16_t omega_dot;
71     int16_t M_0;
72     int16_t a_f0;
73     int16_t a_f1;
74     uint8_t E5b_HS;
75     uint8_t E1B_HS;
76 }almanac[36] ;
77
78 struct WordT1{
79     //Efemeridy 1/4
80      uint16_t IODnav;
81      uint16_t toe;
82      int32_t Mo;
83      uint32_t e;
84      uint32_t sqrtA;
85      } WordType1[36], temporary_WordType1[36];
86
87 struct WordT2{
88     //Efemeridy 2/4
```

```
89        uint16_t IODnav;
90        int32_t OMEGA_0;
91        int32_t io;
92        int32_t omega;
93        int16_t ii;
94        } WordType2[36], temporary_WordType2[36];
95
96  struct WordT3{
97      //Efemeridy 3/4
98      uint16_t IODnav;
99      int32_t omega_dot;
100     int16_t delta_n;
101     int16_t C_uc;
102     int16_t C_us;
103     int16_t C_rc;
104     int16_t C_rs;
105     uint8_t SISA;
106     } WordType3[36], temporary_WordType3[36];
107
108 struct WordT4{
109     uint16_t IODnav;
110     uint8_t SVID;
111     //Efemeridy 4/4
112     int16_t C_ic;
113     int16_t C_is;
114     //casove korekcni parametry
115     uint16_t t_0c;
116     int32_t a_f0;
117     int32_t a_f1;
118     int8_t a_f2;
119     } WordType4[36], temporary_WordType4[36];
120
121 struct WordT5{
122     //Ionosfericke korekcni cleny:
123     uint16_t a_i0;
124     int16_t a_i1;
125     int16_t a_i2;
126     uint8_t Region1;
127     uint8_t Region2;
128     uint8_t Region3;
129     uint8_t Region4;
130     uint8_t Region5;
131     //BGD a validata dat a signalu
132     int16_t BGDa;
133     int16_t BGDb;
134     uint8_t E5b_Hs;
135     uint8_t E1B_HS;
136     uint8_t E5b_DVs;
137     uint8_t E1B_DVs;
138     uint16_t WN;    //GST - Week number
139     uint32_t TOW;   //GST - Time of week
140     } WordType5[36], temporary_WordType5[36];
```

```
141
142 struct WordT6{
143     //GST-UTC konverzni cleny
144     int32_t A0;
145     int32_t A1;
146     int8_t deltat_LS;
147     uint8_t t_ot;
148     uint8_t WN_0t;
149     uint8_t WN_LSF;
150     uint8_t DN;
151     int8_t deltat_LSF;
152     uint32_t TOW;
153     } WordType6[36], temporary_WordType6[36];
154
155 struct WordT7{
156     uint8_t IODa;
157     uint8_t WNa;
158     uint16_t t_0a;
159     //ALMANACH
160     //SV_SVID1 1/2
161     uint8_t SVID1;
162     int16_t delta_sqrtA;
163     uint16_t e;
164     int16_t omega;
165     int16_t d_i;
166     int16_t omega_0;
167     int16_t omega_dot;
168     int16_t M_0;
169     } WordType7[36], temporary_WordType7[36];
170
171 struct WordT8{
172     //SV_SVID1 2/2
173     uint8_t IODa;
174     int16_t a_f0;
175     int16_t a_f1;
176     uint8_t E5b_HS;
177     uint8_t E1B_HS;
178     //SV_SVID2 1/2
179     uint8_t SVID2;
180     int16_t delta_sqrtA;
181     uint16_t e;
182     int16_t omega;
183     int16_t d_i;
184     int16_t omega_0;
185     int16_t omega_dot;
186     } WordType8[36], temporary_WordType8[36];
187
188 struct WordT9{
189     uint8_t IODa;
190     uint8_t WNa;
191     uint16_t t_0a;
192     //SV_SVID2 2/2
```

```
193     int16_t M_0;
194     int16_t a_f0;
195     int16_t a_f1;
196     uint8_t E5b_HS;
197     uint8_t E1B_HS;
198     //SV_SVID3 1/2
199     uint8_t SVID3;
200     int16_t delta_sqrtA;
201     uint16_t e;
202     int16_t omega;
203     int16_t d_i;
204     } WordType9[36], temporary_WordType9[36];
205
206 struct WordT10{
207     uint8_t IODa;
208     //SV_SVID3 2/2
209     int16_t omega_0;
210     int16_t omega_dot;
211     int16_t M_0;
212     int16_t a_f0;
213     int16_t a_f1;
214     uint8_t E5b_HS;
215     uint8_t E1B_HS;
216     //GST-GPS konverzni cleny
217     int16_t A_0G;
218     int16_t A_1G;
219     uint8_t t_0G;
220     uint8_t WN_0G;
221     } WordType10[36], temporary_WordType10[36];
222
223 struct WordT0{
224     uint8_t Time; //validita casu
225     uint16_t WN; //GST - week number
226     uint32_t TOW; //GST - time of week
227     } WordType0[36], temporary_WordType0[36];
228 #endif // PROCESS_GALILEO_DATA_H_
```

B.2 Soubor zdrojového kódu algoritmů

: process_GALILEO_data.c

```
1  /** Author: Tomas Vankat */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stddef.h>
5  #include <inttypes.h>
6  #include <math.h>
7  #include "process_GALILEO_data.h"
8  //vyber frekvence, kvuli rozdilu vypoctu BGD
9  #define E1
10 //#define E5b
11
12 int process_GALILEO_data(uint64_t * buffer, uint8_t SV_number){
13     uint8_t Stranka[240]; //pole pro celou dekodovanou stranku
14     int i,typeword,k;
15     uint64_t nav_msg[16]; //pro kopii bufferu
16
17     if(initialization==0){ //pocatecni vynulovani promennych
18         GST[SV_number-1].time=0;
19         GST[SV_number-1].precise_time=0;
20         GST[SV_number-1].week=0;
21         for( i=0 ; i<36 ; i++){
22             WordType1[i].IODnav=0;
23             WordType1[i].toe=0;
24             WordType1[i].Mo=0;
25             WordType1[i].e=0;
26             WordType1[i].sqrtA=0;
27             WordType2[i].IODnav=0;
28             WordType2[i].OMEGA_0=0;
29             WordType2[i].io=0;
30             WordType2[i].omega=0;
31             WordType2[i].ii=0;
32             WordType3[i].IODnav=0;
33             WordType3[i].omega_dot=0;
34             WordType3[i].delta_n=0;
35             WordType3[i].C_uc=0;
36             WordType3[i].C_us=0;
37             WordType3[i].C_rc=0;
38             WordType3[i].C_rs=0;
39             WordType3[i].SISA=0;
40             WordType4[i].IODnav=0;
41             WordType4[i].SVID=0;
42             WordType4[i].C_ic=0;
43             WordType4[i].C_is=0;
44             WordType4[i].t_0c=0;
45             WordType4[i].a_f0=0;
46             WordType4[i].a_f1=0;
47             WordType4[i].a_f2=0;
48             WordType5[i].a_i0=0;
49             WordType5[i].a_i1=0;
50             WordType5[i].a_i2=0;
```

```
51                  WordType5[i].Region1=0;
52                  WordType5[i].Region2=0;
53                  WordType5[i].Region3=0;
54                  WordType5[i].Region4=0;
55                  WordType5[i].Region5=0;
56                  WordType5[i].BGDa=0;
57                  WordType5[i].BGDb=0;
58                  WordType5[i].E5b_Hs=0;
59                  WordType5[i].E1B_HS=0;
60                  WordType5[i].E5b_DVs=0;
61                  WordType5[i].E1B_DVs=0;
62                  WordType5[i].WN=0;
63                  WordType5[i].TOW=0;
64                  WordType6[i].A0=0;
65                  WordType6[i].A1=0;
66                  WordType6[i].deltat_LS=0;
67                  WordType6[i].t_ot=0;
68                  WordType6[i].WN_0t=0;
69                  WordType6[i].WN_LSF=0;
70                  WordType6[i].DN=0;
71                  WordType6[i].deltat_LSF=0;
72                  WordType6[i].TOW=0;
73                  WordType7[i].IODa=0;
74                  WordType7[i].WNa=0;
75                  WordType7[i].t_0a=0;
76                  WordType7[i].SVID1=0;
77                  WordType7[i].delta_sqrtA=0;
78                  WordType7[i].e=0;
79                  WordType7[i].omega=0;
80                  WordType7[i].d_i=0;
81                  WordType7[i].omega_0=0;
82                  WordType7[i].omega_dot=0;
83                  WordType7[i].M_0=0;
84                  WordType8[i].IODa=0;
85                  WordType8[i].a_f0=0;
86                  WordType8[i].a_f1=0;
87                  WordType8[i].E5b_HS=0;
88                  WordType8[i].E1B_HS=0;
89                  WordType8[i].SVID2=0;
90                  WordType8[i].delta_sqrtA=0;
91                  WordType8[i].e=0;
92                  WordType8[i].omega=0;
93                  WordType8[i].d_i=0;
94                  WordType8[i].omega_0=0;
95                  WordType8[i].omega_dot=0;
96                  WordType9[i].IODa=0;
97                  WordType9[i].WNa=0;
98                  WordType9[i].t_0a=0;
99                  WordType9[i].M_0=0;
100                 WordType9[i].a_f0=0;
101                 WordType9[i].a_f1=0;
102                 WordType9[i].E5b_HS=0;
```

```
103                 WordType9[i].E1B_HS=0;
104                 WordType9[i].SVID3=0;
105                 WordType9[i].delta_sqrtA=0;
106                 WordType9[i].e=0;
107                 WordType9[i].omega=0;
108                 WordType9[i].d_i=0;
109                 WordType10[i].IODa=0;
110                 WordType10[i].omega_0=0;
111                 WordType10[i].omega_dot=0;
112                 WordType10[i].M_0=0;
113                 WordType10[i].a_f0=0;
114                 WordType10[i].a_f1=0;
115                 WordType10[i].E5b_HS=0;
116                 WordType10[i].E1B_HS=0;
117                 WordType10[i].A_0G=0;
118                 WordType10[i].A_1G=0;
119                 WordType10[i].t_0G=0;
120                 WordType10[i].WN_0G=0;
121                 WordType0[i].Time=0;
122                 WordType0[i].WN=0;
123                 WordType0[i].TOW=0;
124             }
125         initialization=1;
126         }
127         GST[SV_number-1].time=GST[SV_number-1].time+0.004;
128             //inkrementace casu druzice
129         GST[SV_number-1].precise_time=GST[SV_number-1].precise_time
                +4000000000; //inkrementace casu druzice
130         for( i=0 ; i<240 ; i++){
131             Stranka[i]=2;
132         }
133         for(i=0;i<16;i++){
134             nav_msg[i]=buffer[i]; //zkopirovani bufferu
135         }
136         k=Dekodovani(nav_msg,Stranka);
137         if(k!=1){
138             return -6; //nenalezen zacatek stranky
139         }
140         typeword=0;//prepsani Typeword z pole do samostatne promenne
141         for( i=2 ; i<8 ; i++ ){
142             if(Stranka[i]>1){
143                 return -5;//nebyl prirazen nit
144             }
145             typeword=typeword<<1;
146             if(Stranka[i]==1){
147                 typeword++;
148             }
149         }
150         if((typeword<0)||(typeword>10)){
151             return -4;//chyba nebo rezervovane dat. slovo
152         }
153         if(CRC(Stranka)!=0){
```

```
154             return -3;//CRC se nevyvulovalo - v datech je chyba
155         }
156         switch (typeword){//zpracovani podle typeword
157         case 1:
158             sortWT1(Stranka,SV_number);
159             break;
160         case 2:
161             sortWT2(Stranka,SV_number);
162             break;
163         case 3:
164             sortWT3(Stranka,SV_number);
165             break;
166         case 4:
167             sortWT4(Stranka,SV_number);
168             break;
169         case 5: //obsahuje GST
170             sortWT5(Stranka,SV_number);
171             GST[SV_number-1].time=WordType5[SV_number-1].TOW +
                    (260*0.004);
172             GST[SV_number-1].precise_time=WordType5[SV_number-1].TOW
                    *1000000000000 + 1040000000000;//(260*4000000000);
173             GST[SV_number-1].week=WordType5[SV_number-1].WN;
174             break;
175         case 6: //obsahuje TOW
176             sortWT6(Stranka,SV_number);
177             GST[SV_number-1].time=WordType6[SV_number-1].TOW +
                    (260*0.004);
178             GST[SV_number-1].precise_time=WordType6[SV_number-1].TOW
                    *1000000000000 + 1040000000000;//(260*4000000000);
179             break;
180         case 7:
181             sortWT7(Stranka,SV_number);
182             break;
183         case 8:
184             sortWT8(Stranka,SV_number);
185             break;
186         case 9:
187             sortWT9(Stranka,SV_number);
188             break;
189         case 10:
190             sortWT10(Stranka,SV_number);
191             break;
192         case 0: //obsahuje GST
193             sortWT0(Stranka,SV_number);
194             GST[SV_number-1].time=WordType0[SV_number-1].TOW +
                    (260*0.004);
195             GST[SV_number-1].precise_time=WordType0[SV_number-1].TOW
                    *1000000000000 + 1040000000000;//(260*4000000000);
196             GST[SV_number-1].week=WordType0[SV_number-1].WN;
197             break;
198         default:
199             break;
```

```
200     }
201     store_almanac_data(SV_number,typeword);//ulozeni dat
            almanachu
202     return typeword;
203 }
204
205 double User_pos(SVID){//nastin funkce vypoctu polohy
206     SV_coordinates(SVID);
207     system_rotation(SVID, 0.085); //1 odhad 85ms
208     /*prange_calc();
209     while((fabs(prev_prange-prange) > tolerance)){
210         receiver_pos();
211         system_rotation(SVID, time0[SVID-1]);
212         Iono_corr();
213         Tropo_corr();
214         prange_corr();
215     }
216     receiver_pos();*/
217     return 0;
218 }
219
220 void system_rotation(uint8_t SVID, double spr_time){ //spr_time
        .. doba sireni signalu mezi vyslanim a prijmem
221     double theta = 0;
222     double X,Y,Z;
223     theta = omega_e * (((double)GST[SVID-1].precise_time
            /1000000000000+spr_time) - ((double)GST[SVID-1].
            precise_time/1000000000000));
224     X = (cos(theta)*SV_coor[SVID-1].X) + (sin(theta)*SV_coor[SVID
            -1].Y); // + 0*SVcoordin[SVID-1].Z;
225     Y = (-sin(theta)*SV_coor[SVID-1].X) + (cos(theta)*SV_coor[
            SVID-1].Y); // + 0*SVcoordin[SVID-1].Z;
226     Z = 1*SV_coor[SVID-1].Z; // + 0*SVcoordin[SVID-1].X + 0*
            SVcoordin[SVID-1].Y
227     SV_coor[SVID-1].X=X;
228     SV_coor[SVID-1].Y=Y;
229     SV_coor[SVID-1].Z=Z;
230 }
231
232 double GST_UTC(uint8_t SVID){
233     double t_E=GST[SVID-1].time;
234     double delta_t_utc,W;
235     //case a - serizovany cas neni v minulosti a nejsme v rozmezi
            +-6hodin od serizeni
236     if ( ((GST[SVID-1].week%256) < WordType6[SVID-1].WN_LSF) ||
            /*bud je aktualni tyden mensi*/
237         ( ((GST[SVID-1].week%256) == WordType6[SVID-1].WN_LSF) &&
                ((((int)GST[SVID-1].time/86400)+1) < WordType6[SVID
                -1].DN)) || /*nebo je tyden stejny a den mensi*/
238         ( ((GST[SVID-1].week%256) == WordType6[SVID-1].WN_LSF) &&
                ((((int)GST[SVID-1].time/86400)+1) == WordType6[SVID
                -1].DN) && (((fmod(GST[SVID-1].time,86400))/3600) < 18
```

```
                )) ) /*nebo je tyden i den stejny a jsme vic jak 6
                hodin pred*/
239         {
240             delta_t_utc = WordType6[SVID-1].deltat_LS + WordType6[
                    SVID-1].A0*pow(2,-30) + WordType6[SVID-1].A1*pow
                    (2,-50)*(t_E - WordType6[SVID-1].t_ot*3600 + 604800*(
                    WordType5[SVID-1].WN - WordType6[SVID-1].WN_0t));
241             UTC.daytime=fmod((t_E - delta_t_utc),86400);
242             return UTC.daytime;
243         }
244         //case b - jsme v rozmezi +-6 hodin od serizeni
245         if ( (((GST[SVID-1].week%256) == WordType6[SVID-1].WN_LSF) &&
                ((((int)GST[SVID-1].time/86400)+1) == WordType6[SVID-1].
                DN) && (((fmod(GST[SVID-1].time,86400))/3600) >= 18)) ||
                /*bud je tyden i den stejny a cas min nez 6 hodin ke konci
                dne*/
246             (((GST[SVID-1].week%256) == WordType6[SVID-1].WN_LSF) &&
                    ((((int)GST[SVID-1].time/86400)+1) == (WordType6[SVID
                    -1].DN+1)) && (((fmod(GST[SVID-1].time,86400))/3600)
                    <= 6)) || /*nebo tyden stejny a den prave vetsi o 1 a
                    cas mensi nez 6. hodina*/
247             (((GST[SVID-1].week%256) == (WordType6[SVID-1].WN_LSF+1))
                    && ((((int)GST[SVID-1].time/86400)+1) == 1) && (((
                    fmod(GST[SVID-1].time,86400))/3600) <= 6)) ) /*nebo je
                    tyden o 1 vetsi a den je prvni v tydnu a cas mensi
                    nez 6 hodin*/
248         {
249             delta_t_utc = WordType6[SVID-1].deltat_LS + WordType6[
                    SVID-1].A0*pow(2,-30) + WordType6[SVID-1].A1*pow
                    (2,-50)*(t_E - WordType6[SVID-1].t_ot*3600 + 604800*(
                    WordType5[SVID-1].WN - WordType6[SVID-1].WN_0t));
250             W =(fmod( ( t_E - delta_t_utc - 43200) , 86400) + 43200 )
                    ;
251             UTC.daytime =fmod(W,( 86400 + WordType6[SVID-1].
                    deltat_LSF - WordType6[SVID-1].deltat_LS ));
252             return UTC.daytime;
253         }
254         //case c - serizovany cas je v minulosti (uz probehl) a
                nejsme v rozmezi 6hodin od serizeni
255         if ( ((GST[SVID-1].week%256) > (WordType6[SVID-1].WN_LSF+1))
                || /*bud je aktualni tyden vetsi o vic nez 1 nez tyden kdy
                se aplikuje leap second*/
256             ( ((GST[SVID-1].week%256) == (WordType6[SVID-1].WN_LSF+1)
                    ) && ((((int)GST[SVID-1].time/86400)+1) > 1)) || /*
                    nebo je tyden o 1 vetsi a den neni prvni v tydnu*/
257             ( ((GST[SVID-1].week%256) == (WordType6[SVID-1].WN_LSF+1)
                    ) && ((((int)GST[SVID-1].time/86400)+1) == 1) && (((
                    fmod(GST[SVID-1].time,86400))/3600) > 6 )) || /*nebo
                    je tyden vetsi o 1 a den 1. v tydnu a cas vetsi nez 6
                    rano*/
258             ( ((GST[SVID-1].week%256) == (WordType6[SVID-1].WN_LSF))
                    && ((((int)GST[SVID-1].time/86400)+1) > (WordType6[
```

```
                SVID-1].DN+1))) || /*nebo je tyden stejny den vetsi o
                vice nez 1*/
259             ( ((GST[SVID-1].week%256) == (WordType6[SVID-1].WN_LSF))
                && ((((int)GST[SVID-1].time/86400)+1) == (WordType6[
                SVID-1].DN+1)) && (((fmod(GST[SVID-1].time,86400))
                /3600) > 6) ) ) /*nebo je tyden stejny den vetsi o 1 a
                 jsem vic jak 6 hodin po*/
260         {
261             delta_t_utc = WordType6[SVID-1].deltat_LSF + WordType6[
                    SVID-1].A0*pow(2,-30) + WordType6[SVID-1].A1*pow
                    (2,-50)*( t_E - WordType6[SVID-1].t_ot*3600 + 604800*(
                    WordType5[SVID-1].WN - WordType6[SVID-1].WN_0t));
262             UTC.daytime =fmod(( t_E - delta_t_utc ), 86400);
263             return UTC.daytime;
264         }
265         return -1;
266     }
267
268     double SV_coordinates(uint8_t SVID){
269         double A,n0,tk,n,M,v,delta_u,delta_r,delta_i,u,r,i,x_,y_,
                Omega,Phi,E,prevE,prevtk;
270         int kep_iter=0,tk_iter=0;
271         if((WordType1[SVID-1].IODnav==WordType2[SVID-1].IODnav) && (
                WordType2[SVID-1].IODnav==WordType3[SVID-1].IODnav) && (
                WordType3[SVID-1].IODnav==WordType4[SVID-1].IODnav)){
272             //IOD vsech efemerid se rovnaji
273             if(WordType1[SVID-1].sqrtA==0){
274                 //nejsou data
275                 SV_coor[SVID-1].X=0;
276                 SV_coor[SVID-1].Y=0;
277                 SV_coor[SVID-1].Z=0;
278                 return -2;
279             }
280         }else{ //IOD efemerid se nerovnaji
281             SV_coor[SVID-1].X=0;
282             SV_coor[SVID-1].Y=0;
283             SV_coor[SVID-1].Z=0;
284             return -1;
285         }
286         A=pow((WordType1[SVID-1].sqrtA*pow(2,-19)),2);
287         n0=sqrt((my/(pow(A,3))));
288         n=n0+(PI*WordType3[SVID-1].delta_n*pow(2,-43));
289         tk=(double)GST[SVID-1].precise_time/1000000000000-60*
                WordType1[SVID-1].toe;
290         while((fabs(prevtk-tk)) > 0.000000001){
291             M=PI*WordType1[SVID-1].Mo*pow(2,-31)+n*tk;
292             E=M;
293             kep_iter=0;
294             while((fabs((prevE-E)) > 0.0000000001)){
295                 prevE=E;
296                 E=M+WordType1[SVID-1].e*pow(2,-33)*sin(prevE);
297                 kep_iter++;
```

```
298                 if(kep_iter>4){ //max 5 iteraci
299                     break;
300                 }
301             }
302             prevtk=tk;
303             tk=(double)GST[SVID-1].precise_time/1000000000000-
                    Time_Correction(SVID,E)-60*WordType1[SVID-1].toe;
304             tk_iter++;
305             if(tk_iter>4){ //max 5 iteraci
306                     break;
307                 }
308             }
309         v=atan2(((sqrt(1-pow(WordType1[SVID-1].e*pow(2,-33),2))*sin(E
                ))/(1-WordType1[SVID-1].e*pow(2,-33)*cos(E))) , ((cos(E)-
                WordType1[SVID-1].e*pow(2,-33))/(1-WordType1[SVID-1].e*pow
                (2,-33)*cos(E))));
310         Phi= v + PI*WordType2[SVID-1].omega*pow(2,-31);
311         delta_u=WordType3[SVID-1].C_us*pow(2,-29)*sin(2*Phi) +
                WordType3[SVID-1].C_uc*pow(2,-29)*cos(2*Phi);
312         delta_r=WordType3[SVID-1].C_rs*pow(2,-5)*sin(2*Phi) +
                WordType3[SVID-1].C_rc*pow(2,-5)*cos(2*Phi);
313         delta_i=WordType4[SVID-1].C_is*pow(2,-29)*sin(2*Phi) +
                WordType4[SVID-1].C_ic*pow(2,-29)*cos(2*Phi);
314         u=Phi+delta_u;
315         r=A*(1-WordType1[SVID-1].e*pow(2,-33)*cos(E))+delta_r;
316         i=PI*WordType2[SVID-1].io*pow(2,-31) + delta_i + PI*WordType2
                [SVID-1].ii*pow(2,-43)*tk;
317         x_=r*cos(u);
318         y_=r*sin(u);
319         Omega=PI*WordType2[SVID-1].OMEGA_0*pow(2,-31) + (PI*WordType3
                [SVID-1].omega_dot*pow(2,-43)-omega_e)*tk - omega_e*
                WordType1[SVID-1].toe*60;
320         SV_coor[SVID-1].X=x_*cos(Omega) - y_*cos(i)*sin(Omega);
321         SV_coor[SVID-1].Y=x_*sin(Omega) + y_*cos(i)*cos(Omega);
322         SV_coor[SVID-1].Z=y_*sin(i);
323         return E;
324 }
325
326 void store_almanac_data(uint8_t SVID, uint8_t typeword){
327 //ulozeni dat z almanachu do struktury dane druzice
328 if(typeword==7 || typeword==8){
329     if( WordType7[SVID-1].SVID1>0 && WordType7[SVID-1].SVID1<37){
330         if(WordType7[SVID-1].IODa == WordType8[SVID-1].IODa){
331             almanac[WordType7[SVID-1].SVID1-1].IODa=WordType7[
                    SVID-1].IODa;
332             almanac[WordType7[SVID-1].SVID1-1].WNa=WordType7[SVID
                    -1].WNa;
333             almanac[WordType7[SVID-1].SVID1-1].t_0a=WordType7[
                    SVID-1].t_0a;
334             almanac[WordType7[SVID-1].SVID1-1].SVID=WordType7[
                    SVID-1].SVID1;
```

```
335                 almanac[WordType7[SVID-1].SVID1-1].delta_sqrtA=
                        WordType7[SVID-1].delta_sqrtA;
336                 almanac[WordType7[SVID-1].SVID1-1].e=WordType7[SVID
                        -1].e;
337                 almanac[WordType7[SVID-1].SVID1-1].omega=WordType7[
                        SVID-1].omega;
338                 almanac[WordType7[SVID-1].SVID1-1].d_i=WordType7[SVID
                        -1].d_i;
339                 almanac[WordType7[SVID-1].SVID1-1].omega_0=WordType7[
                        SVID-1].omega_0;
340                 almanac[WordType7[SVID-1].SVID1-1].omega_dot=
                        WordType7[SVID-1].omega_dot;
341                 almanac[WordType7[SVID-1].SVID1-1].M_0=WordType7[SVID
                        -1].M_0;
342                 almanac[WordType7[SVID-1].SVID1-1].a_f0=WordType8[
                        SVID-1].a_f0;
343                 almanac[WordType7[SVID-1].SVID1-1].a_f1=WordType8[
                        SVID-1].a_f1;
344                 almanac[WordType7[SVID-1].SVID1-1].E5b_HS=WordType8[
                        SVID-1].E5b_HS;
345                 almanac[WordType7[SVID-1].SVID1-1].E1B_HS=WordType8[
                        SVID-1].E1B_HS;
346             }
347         }
348 }
349 if(typeword==8 || typeword==9){
350     if(WordType8[SVID-1].SVID2>0 && WordType8[SVID-1].SVID2<37){
351         if(WordType8[SVID-1].IODa == WordType9[SVID-1].IODa){
352             almanac[WordType8[SVID-1].SVID2-1].IODa=WordType8[
                        SVID-1].IODa;
353             almanac[WordType8[SVID-1].SVID2-1].WNa=WordType9[SVID
                        -1].WNa;
354             almanac[WordType8[SVID-1].SVID2-1].t_0a=WordType9[
                        SVID-1].t_0a;
355             almanac[WordType8[SVID-1].SVID2-1].SVID=WordType8[
                        SVID-1].SVID2;
356             almanac[WordType8[SVID-1].SVID2-1].delta_sqrtA=
                        WordType8[SVID-1].delta_sqrtA;
357             almanac[WordType8[SVID-1].SVID2-1].e=WordType8[SVID
                        -1].e;
358             almanac[WordType8[SVID-1].SVID2-1].omega=WordType8[
                        SVID-1].omega;
359             almanac[WordType8[SVID-1].SVID2-1].d_i=WordType8[SVID
                        -1].d_i;
360             almanac[WordType8[SVID-1].SVID2-1].omega_0=WordType8[
                        SVID-1].omega_0;
361             almanac[WordType8[SVID-1].SVID2-1].omega_dot=
                        WordType8[SVID-1].omega_dot;
362             almanac[WordType8[SVID-1].SVID2-1].M_0=WordType9[SVID
                        -1].M_0;
363             almanac[WordType8[SVID-1].SVID2-1].a_f0=WordType9[
                        SVID-1].a_f0;
```

```
364                 almanac[WordType8[SVID-1].SVID2-1].a_f1=WordType9[
                        SVID-1].a_f1;
365                 almanac[WordType8[SVID-1].SVID2-1].E5b_HS=WordType9[
                        SVID-1].E5b_HS;
366                 almanac[WordType8[SVID-1].SVID2-1].E1B_HS=WordType9[
                        SVID-1].E1B_HS;
367             }
368         }
369 }
370 if(typeword==9 || typeword==10){
371         if(WordType9[SVID-1].SVID3>0 && WordType9[SVID-1].SVID3<37){
372             if(WordType9[SVID-1].IODa == WordType10[SVID-1].IODa){
373                 almanac[WordType9[SVID-1].SVID3-1].IODa=WordType9[
                        SVID-1].IODa;
374                 almanac[WordType9[SVID-1].SVID3-1].WNa=WordType9[SVID
                        -1].WNa;
375                 almanac[WordType9[SVID-1].SVID3-1].t_0a=WordType9[
                        SVID-1].t_0a;
376                 almanac[WordType9[SVID-1].SVID3-1].SVID=WordType9[
                        SVID-1].SVID3;
377                 almanac[WordType9[SVID-1].SVID3-1].delta_sqrtA=
                        WordType9[SVID-1].delta_sqrtA;
378                 almanac[WordType9[SVID-1].SVID3-1].e=WordType9[SVID
                        -1].e;
379                 almanac[WordType9[SVID-1].SVID3-1].omega=WordType9[
                        SVID-1].omega;
380                 almanac[WordType9[SVID-1].SVID3-1].d_i=WordType9[SVID
                        -1].d_i;
381                 almanac[WordType9[SVID-1].SVID3-1].omega_0=WordType10
                        [SVID-1].omega_0;
382                 almanac[WordType9[SVID-1].SVID3-1].omega_dot=
                        WordType10[SVID-1].omega_dot;
383                 almanac[WordType9[SVID-1].SVID3-1].M_0=WordType10[
                        SVID-1].M_0;
384                 almanac[WordType9[SVID-1].SVID3-1].a_f0=WordType10[
                        SVID-1].a_f0;
385                 almanac[WordType9[SVID-1].SVID3-1].a_f1=WordType10[
                        SVID-1].a_f1;
386                 almanac[WordType9[SVID-1].SVID3-1].E5b_HS=WordType10[
                        SVID-1].E5b_HS;
387                 almanac[WordType9[SVID-1].SVID3-1].E1B_HS=WordType10[
                        SVID-1].E1B_HS;
388             }
389         }
390 }
391 }
392
393 void almanac_SV_coordinates(uint8_t SVID,uint8_t SVIDa,int16_t
        delta_sqrtA,uint16_t e,int16_t delta_i,int16_t omega_0,int16_t
         omega_dot,int16_t omega,int16_t M_0,uint16_t toa,uint8_t WNa)
        {
394     //vypocet polohy druzice z almanachu
```

```
395     double A,n0,tk,n,M,v,u,r,i,x_,y_,Omega,Phi,E,prevE;
396     int kep_iter=0;
397     A=pow( ((delta_sqrtA*pow(2,-9))+sqrt(29600000)) , 2 );
398     n0=sqrt((my/(pow(A,3))));
399     tk=GST[SVID-1].time-600*toa;
400     n=n0;
401     M=PI*M_0*pow(2,-15)+n*tk;
402     E=M;
403     while((fabs((prevE-E)) > 0.000000000001)){
404         prevE=E;
405         E=M+e*pow(2,-16)*sin(prevE);
406         kep_iter++;
407         if(kep_iter>4){ //max 5 iteraci
408             break;
409         }
410     }
411     v=atan2(((sqrt(1-pow(e*pow(2,-16),2))*sin(E))/(1-e*pow(2,-16)
            *cos(E))) , ((cos(E)-e*pow(2,-16))/(1-e*pow(2,-16)*cos(E))
            ));
412     Phi= v + PI*omega*pow(2,-15);
413     u=Phi;
414     r=A*(1-e*pow(2,-16)*cos(E));
415     i=(PI*delta_i*pow(2,-14)+0.9773843811); //0.977384rad = 56deg
416     x_=r*cos(u);
417     y_=r*sin(u);
418     Omega=PI*omega_0*pow(2,-15) + (PI*omega_dot*pow(2,-33)-
            omega_e)*tk - omega_e*toa*600;
419     SV_coor[SVIDa-1].X=x_*cos(Omega) - y_*cos(i)*sin(Omega);
420     SV_coor[SVIDa-1].Y=x_*sin(Omega) + y_*cos(i)*cos(Omega);
421     SV_coor[SVIDa-1].Z=y_*sin(i);
422 }
423
424 double Time_Correction(uint8_t SVID, double E){
425     double delta_tr=0, delta_tsv=0;
426     delta_tr=(-4.442807309*pow(10,-10))*WordType1[SVID-1].e*pow
            (2,-33)*WordType1[SVID-1].sqrtA*pow(2,-19)*sin(E);
427     delta_tsv=WordType4[SVID-1].a_f0*pow(2,-34) + WordType4[SVID
            -1].a_f1*pow(2,-46)*(GST[SVID-1].time-WordType4[SVID-1].
            t_0c*60) + WordType4[SVID-1].a_f2*pow(2,-59)*(pow((GST[
            SVID-1].time-WordType4[SVID-1].t_0c*60),2))+delta_tr;
428     #ifdef E1
429     delta_tsv=delta_tsv-WordType5[SVID-1].BGDb*pow(2,-32);
430     #endif // E1
431     #ifdef E5b
432     delta_tsv=delta_tsv -1.70324619362252223*WordType5[SVID-1].
            BGDb*pow(2,-32);
433     #endif // E5b
434     return delta_tsv;
435 }
436
437 void Deinterleaving(uint8_t Prijato[]){
438 uint8_t i,j,k;
```

```
439 uint8_t B[8][30];
440 k=0;
441 for( i=0 ; i<8 ; i++){
442     for( j=0; j<30 ; j++ , k++){
443         B[i][j]=Prijato[k];
444     }
445 }
446 k=0;
447 for( j=0 ; j<30 ; j++){
448     for( i=0; i<8 ; i++ , k++ ){
449         Prijato[k]=B[i][j];
450     }
451 }
452 }
453
454 void ViterbiDecoding(uint8_t prijato[],uint8_t
        nejpravdepodobnejsi[]){
455 struct {
456     uint64_t Posloupnost[2];
457     uint8_t HW;
458 } cesta[64],pomoc[64];
459 uint64_t nasobek;
460 uint8_t stav,i,HW1,HW2,previous1,previous2,bit;
461 bit=0;
462 cesta[0].HW=0;
463 cesta[0].Posloupnost[0]=0;
464 cesta[0].Posloupnost[1]=0;
465 for( i=1 ; i<64 ; i++){
466     cesta[i].HW=255;
467     cesta[i].Posloupnost[0]=0;
468     cesta[i].Posloupnost[1]=0;
469 }
470 for(i=0; i<240 ; i=i+2){ //vnejsi cyklus pres bity
471     for( stav=0 ; stav<64 ; stav++){
472         pomoc[stav].HW=cesta[stav].HW;
473         pomoc[stav].Posloupnost[0]=cesta[stav].Posloupnost[0];
474         pomoc[stav].Posloupnost[1]=cesta[stav].Posloupnost[1];
475     }
476     for( stav=0 ; stav<64 ; stav++){ //vnitrni cyklus pres stavy
477         if(stav>31){
478             bit=1;
479         }
480         else{
481             bit=0;
482         }
483         previous1=2*(stav-(32*(stav/32)));
484         previous2=2*(stav-(32*(stav/32)))+1;
485         HW1=0;
486         HW2=0;
487         //lze udelat jako (0x1111001 & (bit*64+previous1)) % 2
488         //nebo misto bit*64 udelat bit<<6
489         //if(prijato[i]!=((0x1111001 & (bit*64+previous1))%2)){
```

```
490             //nebo:
491             if(prijato[i]!=((((bit+previous1)&0x01)/0x01)^(((bit+
                    previous1)&0x08)/0x08)^(((bit+previous1)&0x10)/0x10)
                    ^(((bit+previous1)&0x20)/0x20)^(((bit+previous1)&0x40)
                    /0x40))){ //G1
492                 HW1++;
493             }
494             //(prijato[i]!=(!((0x1011011 & (bit*64+previous2))%2))){
495             //nebo:
496             if(prijato[i+1]!=(!((((bit+previous1)&0x01)/0x01)^(((bit+
                    previous1)&0x02)/0x02)^(((bit+previous1)&0x08)/0x08)
                    ^(((bit+previous1)&0x10)/0x10)^(((bit+previous1)&0x40)
                    /0x40)))){ //G2
497                 HW1++;
498             }
499             if(HW1==0)
500                 {HW2=2;}
501             else if(HW1==2)
502                 {HW2=0;}
503             else
504                 {HW2=1;}
505             if( (pomoc[previous1].HW+HW1) >= (pomoc[previous2].HW+HW2
                    ) ){
506                 cesta[stav].HW=pomoc[previous2].HW+HW2;
507                 cesta[stav].Posloupnost[0]=pomoc[previous2].
                        Posloupnost[0];
508                 cesta[stav].Posloupnost[1]=pomoc[previous2].
                        Posloupnost[1];
509             }
510             else{
511                 cesta[stav].HW=pomoc[previous1].HW+HW1;
512                 cesta[stav].Posloupnost[0]=pomoc[previous1].
                        Posloupnost[0];
513                 cesta[stav].Posloupnost[1]=pomoc[previous1].
                        Posloupnost[1];
514             }
515
516             cesta[stav].Posloupnost[1]=cesta[stav].Posloupnost[1]<<1;
517             if((cesta[stav].Posloupnost[0]&0x8000000000000000) > 0){
518                 cesta[stav].Posloupnost[1]=cesta[stav].Posloupnost
                        [1]+1;
519             }
520             cesta[stav].Posloupnost[0]=cesta[stav].Posloupnost[0]<<1;
521             cesta[stav].Posloupnost[0]=cesta[stav].Posloupnost[0]+(
                    stav/32);
522         }
523     }
524     //prepsani dekodovanych bitu do pole
525     nasobek=0x0080000000000000;
526     for( i=0 ; i<120 ; i++)
527     {
528         if(nasobek==0){
```

```
529             nasobek=0x8000000000000000;
530         }
531         if(i<56){
532             if((cesta[0].Posloupnost[1]&nasobek)>0){
533                 nejpravdepodobnejsi[i]=1;
534             }
535             else{
536                 nejpravdepodobnejsi[i]=0;
537             }
538             nasobek=nasobek>>1;
539         }
540         else{
541             if((cesta[0].Posloupnost[0]&nasobek)>0){
542                 nejpravdepodobnejsi[i]=1;
543             }
544             else{
545                 nejpravdepodobnejsi[i]=0;
546             }
547             nasobek=nasobek>>1;
548         }
549     }
550     }
551
552     int Dekodovani(uint64_t nav_msg[], uint8_t Stranka[]){ //
        uplatnuje deinterleaving a dekodovani
553         uint64_t posun;
554         uint8_t Prijato[240];
555         uint8_t Dekodovano[120];
556         int16_t i,k;
557         uint8_t inverze=0;
558         uint8_t zacatek=0;
559             if ((nav_msg[0]&0x00000000000003FF)==0x0000000000000160){
560                 //na zacatku je synch. slovo
561                 if ((((nav_msg[3]&0xFC00000000000000)==(0
                        x8000000000000000))) && (((nav_msg[4])&(0
                        x000000000000000F))==(0x0000000000000005))){
562                     zacatek=1;
563                 }
564             }
565             //zkouska inverze
566             else if ((nav_msg[0]&0x00000000000003FF)==0
                    x000000000000029F){
567                 //na zacatku je synch. slovo
568                 if ((((nav_msg[3]&0xFC00000000000000)==(0
                        x7C00000000000000))) && (((nav_msg[4])&(0
                        x000000000000000F))==(0x000000000000000A))){
569                     zacatek=1;
570                     inverze=1;
571                 }
572             }
573             else{
574                 return -5;//neni zacatek stranky
```

```
575             }
576         if (zacatek==1){
577             posun=0x0000000000000400; //maska
578             k=0;
579             for(i=239 ; i>-1 ; i--){
580                 if ((nav_msg[k]&posun)==0){
581                     if (inverze==0){
582                         Prijato[i]=0;
583                     }
584                     else{
585                         Prijato[i]=1;
586                     }
587                 }
588                 else{
589                     if (inverze==0){
590                         Prijato[i]=1;
591                     }
592                     else{
593                         Prijato[i]=0;
594                     }
595                 }
596                 posun=posun<<1;
597                 if (posun==0){ //jsme na konci radky
598                     posun=0x0000000000000001; //1 se vyhodila uz mimo
                            , takze se da na zacatek
599                     k++; //a pokracujeme o radek vys
600                 }
601             }
602             Deinterleaving(Prijato);
603             ViterbiDecoding(Prijato,Dekodovano);
604             for( i=0 ; i<120 ; i++){
605                 Stranka[i+120]=Dekodovano[i];
606             }
607             if(Dekodovano[0]==1){ //pokud byla jako prvni dekodovana
                    licha cast, tak o blok ''vys'' je suda (prvni)
                    polovina dane stranky a v nasledujicim se dekoduje
608                 posun=0x0000000000000010; //pomocna promenna kterou
                        se nasobi prijata posloupnos, abychom dostali jen
                        ten jeden bit co chceme
609                 k=4;
610                 for(i=239 ; i>-1 ; i--){
611                     if ((nav_msg[k]&posun)==0){
612                         if (inverze==0){
613                             Prijato[i]=0;
614                         }
615                         else{
616                             Prijato[i]=1;
617                         }
618                     }
619                     else{
620                         if (inverze==0){
621                             Prijato[i]=1;
```

```
622                         }
623                         else{
624                             Prijato[i]=0;
625                         }
626                     }
627                     posun=posun<<1;
628                     if (posun==0){ //jsme na konci radky
629                         posun=0x0000000000000001; //1 se vyhodila uz
                                mimo, takze se da na zacaek
630                         k++; //a pokracujeme o radek vys
631                     }
632                 }
633                 Deinterleaving(Prijato);
634                 ViterbiDecoding(Prijato,Dekodovano);
635                 for( i=0 ; i<120 ; i++ ){
636                     Stranka[i]=Dekodovano[i];
637                 }
638             }
639             else{  //posledni prijaty blok je suda tedy prvni cast
640                 return -2;
641             }
642             return 1;
643         }
644         return -3;
645     }
646
647     int CRC(uint8_t Page[]){
648         uint8_t G_x
                []={1,1,0,0,0,0,0,1,1,0,0,1,0,0,1,1,0,0,1,1,1,1,1,0,1,1};
649         uint8_t m_x[220];
650         uint8_t i,j;
651         uint32_t Checksum=0;
652         for( i=0 ; i<114 ; i++){
653             m_x[i]=Page[i];
654         }
655         for ( i=120 ; i<226 ; i++){
656             m_x[i-6]=Page[i];
657         }
658
659         for( i=0 ; i<196 ; i++){
660             if(m_x[i]==0){
661                 continue;
662             }
663             for( j=0 ; j<25 ; j++){
664                 m_x[i+j]=m_x[i+j]^G_x[j];
665             }
666         }
667         for( i=196 ; i<220 ; i++){
668             Checksum=Checksum<<1;
669             if(m_x[i]!=0){
670                 Checksum++;
671             }
```

```
672     }
673     return Checksum;
674 }
675 int readvalue(uint8_t begin, uint8_t len, uint8_t page[], uint8_t
        sign){
676     uint8_t i,beg;
677     int val=0;
678     int32_t pom=0;
679     beg=begin;
680     i=begin;
681     if(sign==1){//sing=1 -> signed; sign=0 -> unsigned
682         beg++;
683         if(page[begin]==1){
684             pom = pow(2,(len-1));
685         }
686     }
687     for( i=beg ; i<(begin+len) ; i++ ){
688         val=val<<1;
689         if(i<114){ //protoze slovo je rozdeleno mezi sudou a
                lichou strankou a mezi castmi slova je tail a dalsi
                dva bity
690             if(page[i]==1){
691                 val++;}
692         }
693         else{
694             if(page[i+8]==1){
695                 val++;}
696         }
697     }
698     return val-pom;
699 }
700 //cteni parametru a ukladani do promennych
701 void sortWT1(uint8_t page[],uint8_t SV_number){
702     WordType1[SV_number-1].IODnav=readvalue(8,10,page,0);
703     WordType1[SV_number-1].toe=readvalue(18,14,page,0);
704     WordType1[SV_number-1].Mo=readvalue(32,32,page,1);
705     WordType1[SV_number-1].e=readvalue(64,32,page,0);
706     WordType1[SV_number-1].sqrtA=readvalue(96,32,page,0);
707 }
708 void sortWT2(uint8_t page[],uint8_t SV_number){
709     WordType2[SV_number-1].IODnav=readvalue(8,10,page,0);
710     WordType2[SV_number-1].OMEGA_0=readvalue(18,32,page,1);
711     WordType2[SV_number-1].io=readvalue(50,32,page,1);
712     WordType2[SV_number-1].omega=readvalue(82,32,page,1);
713     WordType2[SV_number-1].ii=readvalue(114,14,page,1);
714 }
715 void sortWT3(uint8_t page[],uint8_t SV_number){
716     WordType3[SV_number-1].IODnav=readvalue(8,10,page,0);
717     WordType3[SV_number-1].omega_dot=readvalue(18,24,page,1);
718     WordType3[SV_number-1].delta_n=readvalue(42,16,page,1);
719     WordType3[SV_number-1].C_uc=readvalue(58,16,page,1);
720     WordType3[SV_number-1].C_us=readvalue(74,16,page,1);
```

```
721     WordType3[SV_number-1].C_rc=readvalue(90,16,page,1);
722     WordType3[SV_number-1].C_rs=readvalue(106,16,page,1);
723     WordType3[SV_number-1].SISA=readvalue(122,8,page,0);
724 }
725 void sortWT4(uint8_t page[],uint8_t SV_number){
726     WordType4[SV_number-1].IODnav=readvalue(8,10,page,0);
727     WordType4[SV_number-1].SVID=readvalue(18,6,page,0);
728     WordType4[SV_number-1].C_ic=readvalue(24,16,page,1);
729     WordType4[SV_number-1].C_is=readvalue(40,16,page,1);
730     WordType4[SV_number-1].t_0c=readvalue(56,14,page,0);
731     WordType4[SV_number-1].a_f0=readvalue(70,31,page,1);
732     WordType4[SV_number-1].a_f1=readvalue(101,21,page,1);
733     WordType4[SV_number-1].a_f2=readvalue(122,6,page,1);
734 }
735 void sortWT5(uint8_t page[],uint8_t SV_number){
736     WordType5[SV_number-1].a_i0=readvalue(8,11,page,0);
737     WordType5[SV_number-1].a_i1=readvalue(19,11,page,1);
738     WordType5[SV_number-1].a_i2=readvalue(30,14,page,1);
739     WordType5[SV_number-1].Region1=readvalue(44,1,page,0);
740     WordType5[SV_number-1].Region2=readvalue(45,1,page,0);
741     WordType5[SV_number-1].Region3=readvalue(46,1,page,0);
742     WordType5[SV_number-1].Region4=readvalue(47,1,page,0);
743     WordType5[SV_number-1].Region5=readvalue(48,1,page,0);
744     WordType5[SV_number-1].BGDa=readvalue(49,10,page,1);
745     WordType5[SV_number-1].BGDb=readvalue(59,10,page,1);
746     WordType5[SV_number-1].E5b_Hs=readvalue(69,2,page,0);
747     WordType5[SV_number-1].E1B_HS=readvalue(71,2,page,0);
748     WordType5[SV_number-1].E5b_DVs=readvalue(73,1,page,0);
749     WordType5[SV_number-1].E1B_DVs=readvalue(74,1,page,0);
750     WordType5[SV_number-1].WN=readvalue(75,12,page,0);
751     WordType5[SV_number-1].TOW=readvalue(87,20,page,0);
752 }
753 void sortWT6(uint8_t page[],uint8_t SV_number){
754     WordType6[SV_number-1].A0=readvalue(8,32,page,1);
755     WordType6[SV_number-1].A1=readvalue(40,24,page,1);
756     WordType6[SV_number-1].deltat_LS=readvalue(64,8,page,1);
757     WordType6[SV_number-1].t_ot=readvalue(72,8,page,0);
758     WordType6[SV_number-1].WN_0t=readvalue(80,8,page,0);
759     WordType6[SV_number-1].WN_LSF=readvalue(88,8,page,0);
760     WordType6[SV_number-1].DN=readvalue(96,3,page,0);
761     WordType6[SV_number-1].deltat_LSF=readvalue(99,8,page,1);
762     WordType6[SV_number-1].TOW=readvalue(107,20,page,0);
763 }
764 void sortWT7(uint8_t page[],uint8_t SV_number){
765     WordType7[SV_number-1].IODa=readvalue(8,4,page,0);
766     WordType7[SV_number-1].WNa=readvalue(12,2,page,0);
767     WordType7[SV_number-1].t_0a=readvalue(14,10,page,0);
768     WordType7[SV_number-1].SVID1=readvalue(24,6,page,0);
769     WordType7[SV_number-1].delta_sqrtA=readvalue(30,13,page,1);
770     WordType7[SV_number-1].e=readvalue(43,11,page,0);
771     WordType7[SV_number-1].omega=readvalue(54,16,page,1);
772     WordType7[SV_number-1].d_i=readvalue(70,11,page,1);
```

```
773         WordType7[SV_number-1].omega_0=readvalue(81,16,page,1);
774         WordType7[SV_number-1].omega_dot=readvalue(97,11,page,1);
775         WordType7[SV_number-1].M_0=readvalue(108,16,page,1);
776     }
777     void sortWT8(uint8_t page[],uint8_t SV_number){
778         WordType8[SV_number-1].IODa=readvalue(8,4,page,0);
779         WordType8[SV_number-1].a_f0=readvalue(12,16,page,1);
780         WordType8[SV_number-1].a_f1=readvalue(28,13,page,1);
781         WordType8[SV_number-1].E5b_HS=readvalue(41,2,page,0);
782         WordType8[SV_number-1].E1B_HS=readvalue(43,2,page,0);
783         WordType8[SV_number-1].SVID2=readvalue(45,6,page,0);
784         WordType8[SV_number-1].delta_sqrtA=readvalue(51,13,page,1);
785         WordType8[SV_number-1].e=readvalue(64,11,page,0);
786         WordType8[SV_number-1].omega=readvalue(75,16,page,1);
787         WordType8[SV_number-1].d_i=readvalue(91,11,page,1);
788         WordType8[SV_number-1].omega_0=readvalue(102,16,page,1);
789         WordType8[SV_number-1].omega_dot=readvalue(118,11,page,1);
790     }
791     void sortWT9(uint8_t page[],uint8_t SV_number){
792         WordType9[SV_number-1].IODa=readvalue(8,4,page,0);
793         WordType9[SV_number-1].WNa=readvalue(12,2,page,0);
794         WordType9[SV_number-1].t_0a=readvalue(14,10,page,0);
795         WordType9[SV_number-1].M_0=readvalue(24,16,page,1);
796         WordType9[SV_number-1].a_f0=readvalue(40,16,page,1);
797         WordType9[SV_number-1].a_f1=readvalue(56,13,page,1);
798         WordType9[SV_number-1].E5b_HS=readvalue(69,2,page,0);
799         WordType9[SV_number-1].E1B_HS=readvalue(71,2,page,0);
800         WordType9[SV_number-1].SVID3=readvalue(73,6,page,0);
801         WordType9[SV_number-1].delta_sqrtA=readvalue(79,13,page,1);
802         WordType9[SV_number-1].e=readvalue(92,11,page,0);
803         WordType9[SV_number-1].omega=readvalue(103,16,page,1);
804         WordType9[SV_number-1].d_i=readvalue(119,11,page,1);
805     }
806     void sortWT10(uint8_t page[],uint8_t SV_number){
807         WordType10[SV_number-1].IODa=readvalue(8,4,page,0);
808         WordType10[SV_number-1].omega_0=readvalue(12,16,page,1);
809         WordType10[SV_number-1].omega_dot=readvalue(28,11,page,1);
810         WordType10[SV_number-1].M_0=readvalue(39,16,page,1);
811         WordType10[SV_number-1].a_f0=readvalue(55,16,page,1);
812         WordType10[SV_number-1].a_f1=readvalue(71,13,page,1);
813         WordType10[SV_number-1].E5b_HS=readvalue(84,2,page,0);
814         WordType10[SV_number-1].E1B_HS=readvalue(86,2,page,0);
815         WordType10[SV_number-1].A_0G=readvalue(88,16,page,1);
816         WordType10[SV_number-1].A_1G=readvalue(104,12,page,1);
817         WordType10[SV_number-1].t_0G=readvalue(116,8,page,0);
818         WordType10[SV_number-1].WN_0G=readvalue(124,6,page,0);
819     }
820     void sortWT0(uint8_t page[],uint8_t SV_number){
821         WordType0[SV_number-1].Time=readvalue(8,2,page,0);
822         WordType0[SV_number-1].WN=readvalue(98,12,page,0);
823         WordType0[SV_number-1].TOW=readvalue(110,20,page,0);
824     }
```