ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra elektromagnetického pole

Algoritmy pro zpracování navigační zprávy systému GPS

The GPS Navigation Message Processing Algorithms

Bakalářská práce

Studijní program: Komunikace, multimédia a elektronika

Studijní obor: Komunikační technika

Vedoucí práce: prof. Ing. František Vejražka, CSc.

Michaela Tomanová

Praha, květen 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tomanová** Jméno: **Michaela** Osobní číslo: **434808**

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:

Algoritmy pro zpracování navigační zprávy systému GPS

Název bakalářské práce anglicky:

The GPS Navigation Message Processing Algorithms

Pokyny pro vypracování:

Navrhněte algoritmus nalezení preambule v navigační zprávě systému GPS. Navrhněte postup čtení dat ze zprávy včetně dekódování FEC (Forward Error Corrections).
Připravte si postup čtení zprávy a výstupu efemerid družic ze vzorku experimentálního demonstrátoru ?Macek?.
Písemná zpráva: do 50 stran, příp. pravidelné dílčí zprávy.

Seznam doporučené literatury:

[1] ICD dokument GPS L1 n- bude upřesněno
[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: **25.01.2017** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **25.05.2018**

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna 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 studentky

Prohlášení

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

Praha, 26.5.2017

……

Michaela Tomanová

Poděkování

V prvé řadě bych velmi ráda poděkovala panu profesoru Vejražkovi za čas, který věnoval mé bakalářské práci, ať už v podobě konzultací k tématu, kontrole a opravě obsahu práce či doporučování vhodné literatury.

Dále bych též chtěla poděkovat studentům doktorandského studia Jiřímu Svatoňovi za pomoc s testováním programu, Václavu Navrátilovi za konzultace k tématu navigační zprávy a Zdeňku Fialovi za zodpovězení mých dotazů ohledně programování v programovacím jazyku C.

Abstrakt

Cílem bakalářské práce je navrhnout algoritmus pro extrakci dat z navigační zprávy systému GPS, implementovat tento algoritmus do prototypu experimentálního multikonstelačního přijímače a ověřit jeho funkčnost. V práci je nejprve uveden teoretický rozbor navigační zprávy systému GPS, tj. rozložení, obsah a kódování navigační zprávy. Následně je popsána realizace programu včetně vývojového diagramu a popisu jednotlivých kroků. V závěru je shrnuto testování programu na reálném signálu obsahujícím navigační zprávu. Samotný program napsaný v programovacím jazyku C je přiložen k bakalářské práci. Mimo navigační zprávy systému GPS se práce okrajově zabývá i zprávou systému BeiDou. Opět je uveden teoretický rozbor navigační zprávy, který je ověřen výpočtem ze zkušebních dat, a navrhovaný postup pro realizaci programu pro zpracování navigační zprávy systému BeiDou.

Klíčová slova: Hammingův kód, navigační zpráva systému GPS, navigační zpráva systému BeiDou

Abstract

The goal of this bachelor thesis is to project an algorithm for GPS navigation message processing, to implement this algorithm to an experimental multi-constellation receiver prototype and to verify its functionality. First, a GPS navigation message theoretical analysis is given; i.e. the structure, content and encoding of the message. After that, the program realization is described; including the flow diagram and description of particular steps. In the conclusion, the program testing using a real signal which contains the navigation message is summarized. The program written in the C programming language is enclosed. In addition to the GPS navigation message this thesis deals marginally with the BeiDou navigation message. There is given also a BeiDou navigation message theoretical analysis verified by test data calculations and a proposed procedure for a BeiDou navigation message processing program realization.

Keywords: Hamming code, GPS navigation message, BeiDou navigation message

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

1.1 Obecně o navigačních systémech

Globální družicové navigační systémy (GNSS) jsou radiové systémy využívající umělé družice Země, které slouží k určení geografické polohy uživatele a následně k jeho navigování. Realizaci těchto systémů umožnil velký technický pokrok především v oblastech kosmické technologie, mikroelektroniky a počítačových technologií [1].

Hlavními výhodami GNSS je určení polohy uživatele kdekoliv na planetě s velkou přesností, která se s neustálým vývojem zvyšuje, a nepřetržitá činnost systému bez ohledu na denní či noční dobu a počasí. To je důvodem, proč se družicové systémy uplatnily už na počátku své existence, a to především ve vojenství v letecké a námořní navigaci. V dnešní době je služba GNSS poskytnuta i veřejnosti a využívá se krom letecké a námořní navigace i pro navigaci pozemní, též např. v oblasti telekomunikačních služeb a pro rekreační aktivity [1]. Díky navigačním systémům je náš život výrazně zjednodušen, dokonce jsou životy mnoha lidí vlivem rychlé a přesné navigace zachraňovány.

Funkci navigačních systémů zajišťují 3 segmenty

- vesmírný,
- řídicí a
- uživatelský.

Vesmírným segmentem navigačního systému jsou družice obíhající kolem Země, které jsou optimálně rozmístěny tak, aby určená poloha byla co nejpřesnější. Řídící segment je složen z hlavní kontrolní stanice a monitorovacích stanic, které jsou umístěny v oblasti rovníku. Úkolem tohoto segmentu je řízení družic vysíláním příslušné zprávy a kontrola, zda družice např. nevybočují ze své dráhy. V rámci uživatelského či pozemního segmentu dochází k příjmu signálu družice vhodným přijímačem, ke zpracování tohoto signálu a k následnému výpočtu polohy uživatele [2].

Bez výše uvedených segmentů nelze zaručit správné fungování navigačního systému. V dalších záležitostech už se ale jednotlivé systémy mírně liší. Těmito záležitostmi je například počet a umístění družic, kmitočet a modulace družicemi vysílaného signálu, či tvar a kódování navigační zprávy.

1.1.1 GPS

GPS (Global Positioning System) je nejstarším z navigačních systémů, který byl vytvořen Spojenými státy původně především pro vojenské účely. Nominální konstelace systému se skládá z 24 družic, které obíhají Zemi po středních drahách ve výšce přibližně 20 200 km. Vzhledem k celosvětovému významu GPS udržuje USA na oběžných drahách i další (záložní) družice, které v případě poruchy nahradí družice činné. Z téhož důvodu se také snaží o neustálý vývoj a obnovu družic. Současná konstelace je tvořena jak novými, tak i staršími satelity z bloků IIR („Replenishment“), IIR(M) („Modernized“) a IIF („Follow-on“) s předpokládanou životností 7,5 roku, popř. 12 let. Byly navrženy a vyvinuty i satelity GPS III s předpokládanou životností 15 let, které by měly být vypouštěny od roku 2018. Část z nich už byla postavena a otestována. Družice GPS vysílají signály na třech kmitočtech označovaných L1 (1575,42 MHz), L2 (1227,60 MHz) a L5 (1176,45 MHz) [3][4].

Podstatou GPS je stálé pokrytí celého povrchu Země alespoň čtyřmi družicemi, protože k určování polohy dochází měřením vzdálenosti právě alespoň k těmto čtyřem družicím. Poloha

uživatele se pak získá jako průsečík čtyř koulí, jejichž poloměrem je vzdálenost družice od uživatele, a jejichž střed leží ve fázovém středu antény družice.

1.1.2 GLONASS

Dalším perspektivním systémem je ruský navigační systém GLONASS. Ačkoli byl stejně jako GPS vybudován původně pro vojenské použití, je dnes taktéž využíván i civilní veřejností. Nominální konstelaci systému tvoří 24 činných družic obíhajících ve třech rovinách. Kromě toho je na drahách stejně jako v případě GPS umístěno ještě několik dalších družic, které slouží jako záložní. Rusko provozuje zatím 3 druhy družic, původní družice systému GLONASS, modernizované družice typu M a družice typu K. V rámci dalšího vývoje systému jsou plánovány další typy. Družice obíhají ve výšce 19 100 km a jejich doba oběhu činí 11 hodin a 15 minut. Družice pracují na kmitočtech L1 (cca 1,6 GHz) a L2 (cca 1,25 GHz) s kmitočtovým dělením a jsou modulovány modulací BPSK [5].

Systém GLONASS se při svém vývoji setkal s mnohými technickými potížemi. Při startu v prosinci 2010 se zřítily tři družice, v dubnu 2013 došlo při vypouštění k selhání nosné rakety a byly zničeny další tři družice. Kromě toho životnost některých družic, které byly úspěšně umístěny na oběžnou dráhu, byla často jen několik hodin. GLONASS se navíc výrazně odlišuje od ostatních systémů GNSS kmitočtovým dělením signálu. Novější družice poskytují i signály s kódovým dělením, což je výhodnější pro zpracování signálu v přijímači. Tyto družice s kmitočtovým i kódovým dělením byly označeny jako GLONASS typu K a jsou vypouštěny od roku 2011 [3].

1.1.3 Galileo

Evropský systém Galileo je pravděpodobně prvním navigačním systémem, který není primárně navržen pro vojenské účely, ale je už od projektové fáze řízený civilní správou. V plném provozu by měl systém Galileo sestávat ze 30 družic, z nichž by 27 mělo být operačních, které budou obíhat na středních drahách ve třech rovinách ve výšce asi 23 200 km, a 3 družice by byly geostacionární a šířily by signály EGNOS, které zajišťují především integritu systému a diferenční korekce pro systémy Galileo, GPS a GLONASS. Systém bude vysílat 11 navigačních signálů na různých kmitočtech, které budou zabezpečovat služby různé úrovně [6].

Počátky systému Galileo sahají do roku 1994. Tehdy bylo v plánu budování dvou navigačních systémů: GNSS1 a GNSS2. GNSS1 měl být systém zajišťující kompatibilitu systémů GPS a GLONASS a měl tím podpořit myšlenku spojení těchto dvou systémů (tedy využití družic obou systémů), a tím zlepšit kvality navigačních služeb. GNSS2 měl být zcela novým navigačním systémem pracujícím na jiných principech než GPS. Jeho hlavní výhodou mělo být poučení se z chyb při výstavbě ostatních systémů, a tudíž i zpřesnění určování polohy. Nicméně kvůli mezinárodním roztržkám a též z ekonomických důvodů vznikl pouze GNSS2, později označovaný jako Galileo. Původní plán spuštění Galilea v roce 2008 nebyl dodržen a ani dnes není systém zcela funkční, čímž ztrácí výhodu oproti ostatním systémům, které byly během posledních let modernizovány [3].

1.1.4 BeiDou

Čínský navigační systém BeiDou je též známý pod názvem COMPASS. Ačkoli zatím není v plném provozu, jeví se tento systém velmi perspektivně. Konstelace družic je oproti ostatním systémům složitější. Předpokládá se, že se bude systém skládat z celkem 35 družic, tedy z pěti geostacionárních družic (GEO), z 27 družic obíhajících na středních drahách Země (MEO) a ze tří geosynchronních družic s inklinovanými drahami (IGSO). Družice se nacházejí v různých výškách – GEO a IGSO družice ve výšce 35 786 km, zatímco MEO družice ve výšce 21 528 km nad Zemí. Signálů vysílaných systémem BeiDou má být šest, mezi nimi by měl být zahrnut i signál podobný signálu L1C systému GPS. Plného pokrytí by měl systém dosáhnout asi v roce 2020 [3][7].

1.1.5 Ostatní

Ve snaze konkurovat Spojeným státům vznikaly další navigační systémy budované ostatními světovými mocnostmi. Navigační systémy jsou v dnešní době popularizovány natolik, že se o vybudování vlastních systémů snaží už i menší státy.

Jako příklad je možno uvést indický navigační systém IRNSS (Indian Regional Navigation Satellite System). Na rozdíl od výše uvedených systémů je IRNSS pouze regionálním navigačním systémem, který má za úkol poskytnout přesné informace o poloze převážně uživatelům nacházejícím se v Indii či mimo indické území do 1500 km od indických hranic. Systém je složen ze sedmi družic, tři z nich jsou geostacionární a zbylé čtyři jsou umístěny na geosynchronních drahách ve dvou různých rovinách. IRNSS poskytuje 2 druhy služeb, jednu standardní dostupnou všem uživatelům a druhou omezenou, která je kódována a určena jen autorizovaným uživatelům [8].

Za zmínku stojí ještě japonský Quasi-Zenith Satellite System (QZSS) přezdívaný jako „japonská GPS“. I tento systém má být regionální a budou ho tvořit družice na geostacionárních a IGSO drahách. Družice systému QZSS vysílají korekce pro systémy GPS a GLONASS, čímž zvyšují přesnost určení polohy nejen nad územím Japonska, ale i nad značnou částí pacifického oceánu. Pro systém QZSS je typické rozmístění družic tak, aby byla vždy vidět jedna družice s elevací nad 70°, díky čemuž je možné přesné určení polohy i ve vysoké zástavbě [9].

1.2 Předmět bakalářské práce

Předmětem bakalářské práce je návrh algoritmu pro synchronizaci přijímané navigační zprávy systému GPS a pro její zpracování. Algoritmus bude následně implementován do prototypu experimentálního multikonstelačního přijímače.

Podstatou multikonstelačního přijímače je schopnost přijímat signály od družic různých navigačních systémů a následně je zpracovávat. Díky této schopnosti disponuje přijímač vyšší spolehlivostí a přesností určení polohy, je odolnější např. vůči vícecestnému šíření či záměrnému i náhodnému rušení. Funkčnost přijímače neohrozí ani celkový výpadek jednoho navigačního systému.

Tím, že je přijímač schopen zpracovávat signál z více navigačních systémů, má k dispozici více vhodných družic pro výpočet polohy. S rostoucím počtem vhodných družic se snižuje parametr PDOP (Position Dilution Of Precision), který udává, jak se změní chyba v určení polohy v závislosti na rozmístění družic na obloze. Některé nově vznikající systémy navíc myšlenku multikonstelačního přijímače podporují a do svých navigačních zpráv vkládají proměnné popisující vztahy k služebně nejstaršímu systému GPS či k ostatním dříve vyvinutým systémům.

Problematika, kterou se tato bakalářská práce zabývá, je součástí týmového projektu. Ten je rozdělen na jednotlivé dílčí úkoly vedoucí k realizaci přijímače. Těmito úkoly jsou např. příjem vysokofrekvenčního signálu, jeho demodulace, změření zdánlivých vzdáleností od družic, dekódování navigačních zpráv systémů a získání parametrů, které přenášejí, výpočet poloh družic a konečné určení polohy uživatele.

2 Navigační systém GPS

2.1 Navigační zpráva systému GPS

Navigační zpráva je zpráva vysílaná družicemi navigačních systémů, která poskytuje veškerá potřebná data o těchto družicích včetně dat pro vypočet jejich polohy. Kompletní informace nejen o navigační zprávě, ale i konstelaci systému či kódování a modulaci vysílaného signálu lze nalézt v tzv. ICD dokumentu (Interface Control Document) příslušného navigačního systému [10].

2.1.1 Rozložení navigační zprávy GPS

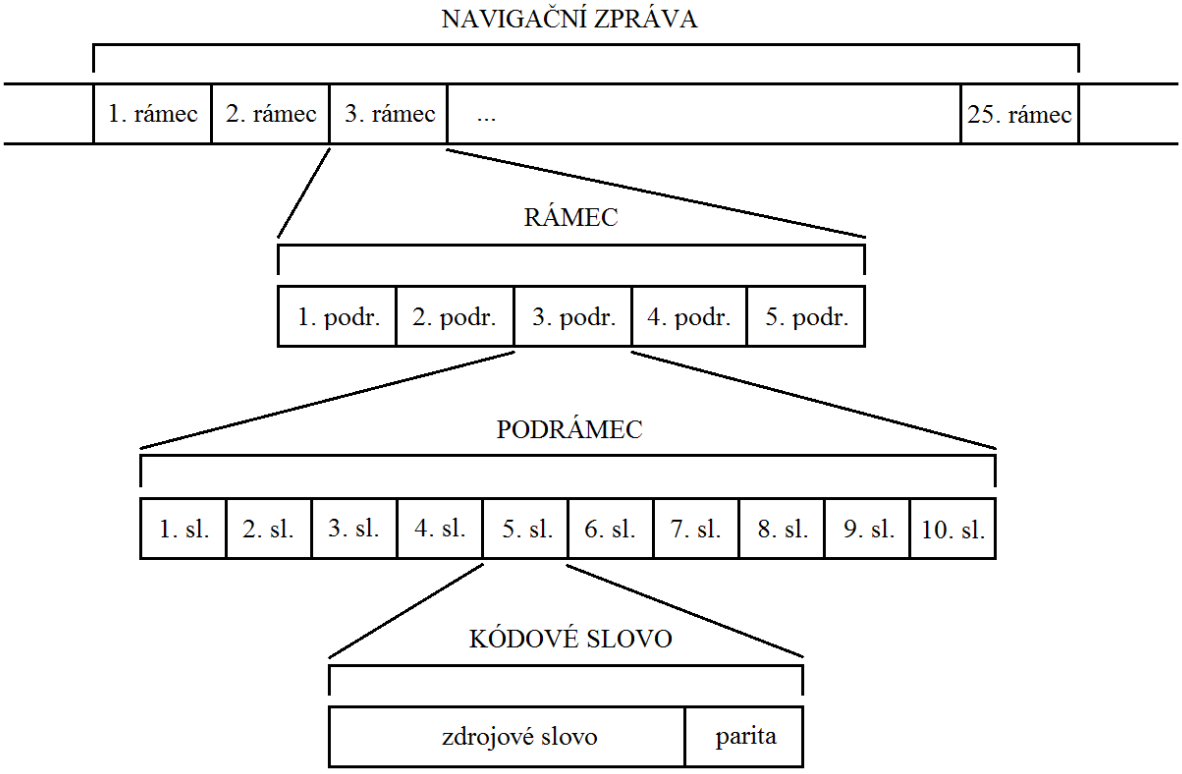
Aby bylo možné se v navigační zprávě dobře orientovat, je tato zpráva postupně dělena na menší části. Nejprve se zpráva dělí do tzv. rámců (angl. frame). Jedna navigační zpráva obsahuje 25 rámců, a každý z těchto rámců je dále dělen do 5 podrámců (angl. subframe). Rozložení proměnných v každém z těchto podrámců je stálé a přesně určené ICD dokumentem.

K nalezení jednotlivých podrámců je potřeba zasynchronizovat přijímač na příchozí data. K tomu slouží tzv. preambule, osmibitové synchronizační slovo, které se vyskytuje na začátku každého podrámce. Synchronizace podrámců je velice důležitá, bez ní nelze navigační zprávu dekódovat.

Každý podrámec obsahuje 10 kódových slov, každé kódové slovo se skládá ze třiceti bitů. Toto dělení slouží především ke kódování, jednotlivá slova jsou kódována zvlášť.

Navigační zpráva se tedy celkem skládá z 37500 bitů. Je vysílána přenosovou rychlostí 50 b/s, z čehož vyplývá, že doba trvání jedné celé navigační zprávy je 12,5 minuty. Zpráva je vysílána nepřetržitě, tj. po jednom 25-rámcovém úseku ihned následuje další [10].

Rozložení navigační zprávy je na obrázku č. 1:

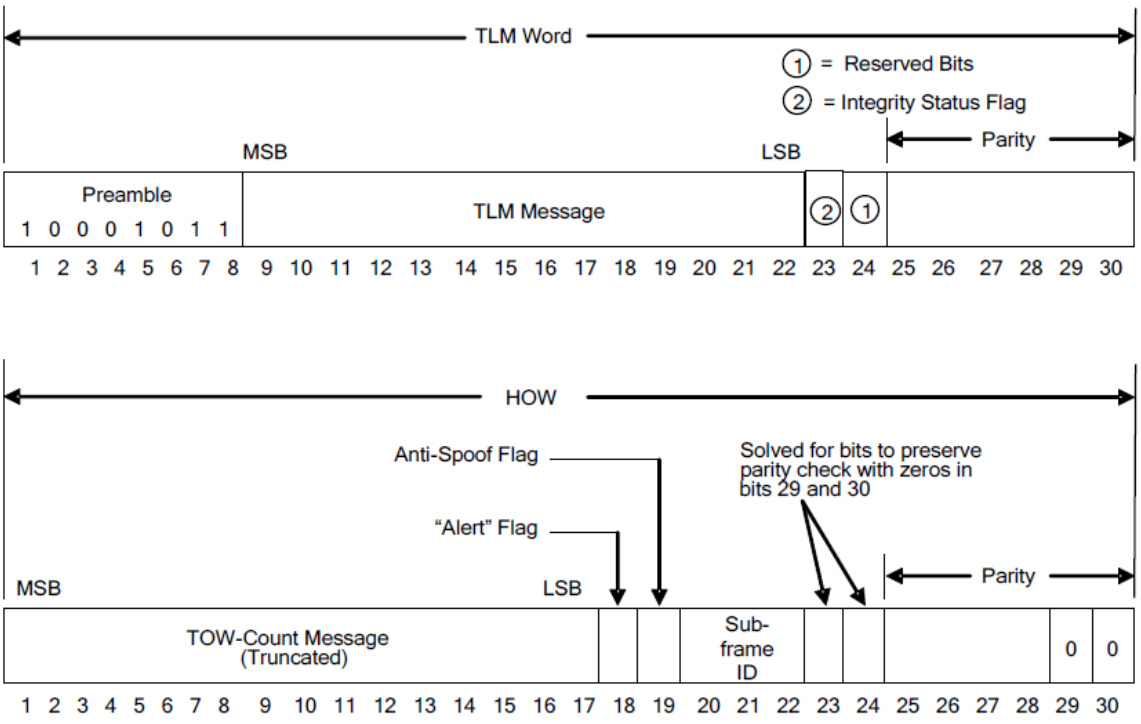


Obrázek 1 - Struktura navigační zprávy systému GPS

2.1.2 Obsah navigační zprávy GPS

Jak již bylo zmíněno dříve, navigační zpráva obsahuje kompletní data potřebná k výpočtu polohy družice. Význam prvních dvou slov v podrámci je u všech podrámců stejný. Prvním slovem je tzv. TLM slovo (angl. Telemetry Word). Na jeho začátku je osmibitová preambule, poté následuje 14-bitová TLM zpráva, která se využívá pro službu PPS. Druhé slovo je označeno jako HOW (angl. Handover Word). Prvních 17 bitů označuje řadové číslo podrámce od půlnoci ze soboty na neděli podle UTC (Coordinated Universal Time). Slovo HOW též obsahuje pořadí podrámce v rámci, které určuje, jak mají být data v podrámci tříděna [10].

Přesné rozložení bitů ve slovech TLM a HOW je na obrázku č. 2, viz [10]:

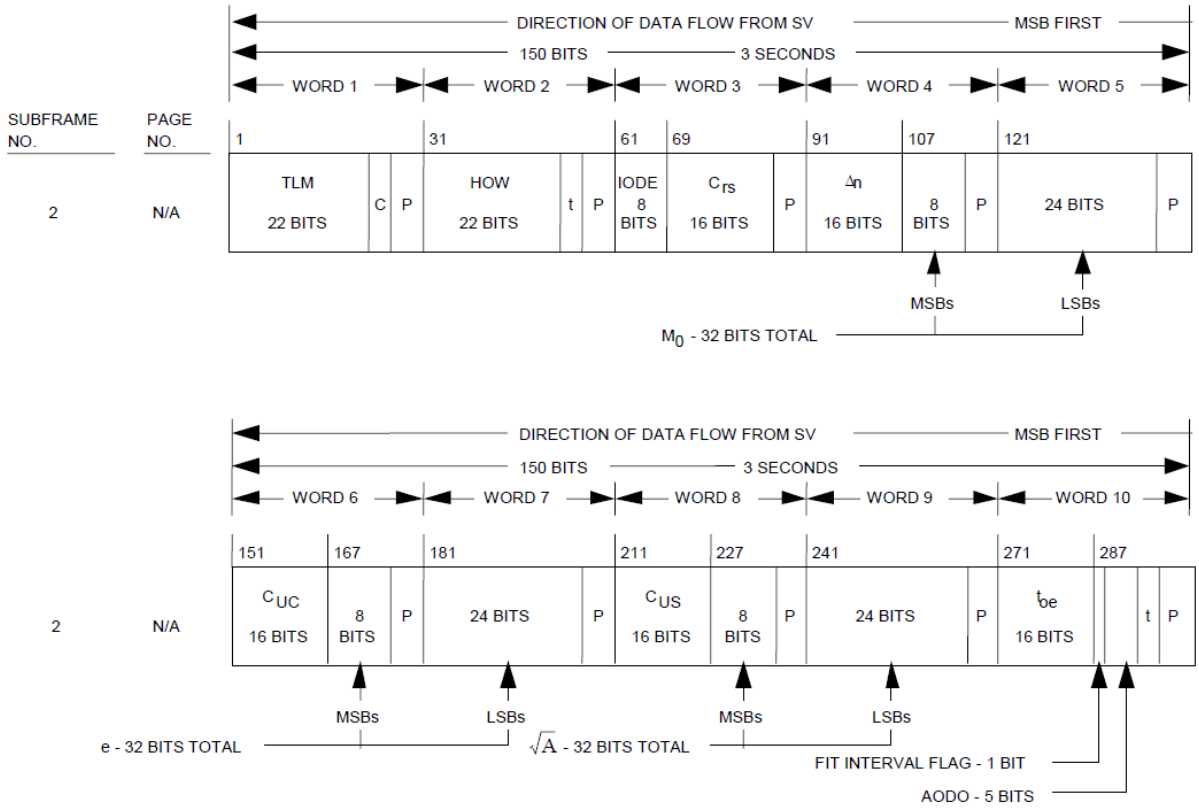


Obrázek 2 - Rozložení bitů ve slovech TLM a HOW v navigační zprávě systému GPS [10]

Data obsažená v jednotlivých podrámcích můžeme rozdělit na tzv. efemeridy, almanachy a další (pomocná) data, která se nacházejí především v prvním podrámci.

Efemeridy jsou podrobné keplerovské parametry dráhy té družice, která vysílá přijímačem přijatou navigační zprávu, a nacházejí se vždy ve druhém a třetím podrámci. Efemeridy obsahují především proměnné pro výpočet polohy této družice a jsou aktualizovány několikrát denně, obvykle jednou za dvě hodiny [2].

Následující obrázek č. 3 zobrazuje rozložení proměnných v druhém podrámci:



Obrázek 3 - Rozložení proměnných ve druhém podrámci navigační zprávy systému GPS [10]

Almanachy obsahují keplerovské parametry drah ostatních družic, které jsou zkrácené na menší počet desetinných míst. Nacházejí se ve čtvrtém a v pátém podrámci. Tyto informace se aktualizují pouze několikrát za týden. Na rozdíl od efemerid záleží na tom, která stránka podrámce byla právě přijata (tj. v kolikátém rámci navigační zprávy se daný podrámec nachází), protože se jednotlivé stránky almanachu významově liší.

Některé stránky obsahují stejné proměnné patřící k různým družicím a mají tak společné rozložení, další jsou zcela zvláštní a zbylé stránky data vůbec neobsahují, bity v nich jsou rezervované pro další použití. Mezi zvláštní stránky patří 25. stránka 4. podrámce, kde je uložena konfigurace družic a stav 25. – 32. družice, dále pak 25. stránka 5. podrámce, kde se nachází stav zbylých družic, tedy 1. – 24. družice, a 18. stránka 4. podrámce, kde jsou uloženy koeficienty tzv. Kloboucharova modelu ionosféry a UTC data [2][10].

Na následujícím obrázku č. 4 je zobrazeno přesné rozložení dat v almanachu:

Subframe	Page(s)	Data
4	1, 6, 11, 16 and 21	Reserved
	2, 3, 4, 5, 7, 8, 9 and 10	almanac data for SV 25 through 32 respectively
	12, 19, 20, 22, 23 and 24	Reserved
	13	NMCT
	14 and 15	Reserved for system use
	17	Special messages
	18	Ionospheric and UTC data
	25	A-S flags/SV configurations for 32 SVs, plus SV health for SV 25 through 32
5	1 through 24	almanac data for SV 1 through 24
	25	SV health data for SV 1 through 24, the almanac reference time, the almanac reference week number

Obrázek 4 - Obsah jednotlivých stránek ve 4. a 5. podrámci navigační zprávy systému GPS [10]

2.1.3 Kódování navigační zprávy systému GPS

2.1.3.1 Obecně o kódování

Signál s navigační zprávou je přenášen komunikačním kanálem s rušením, není tedy možné se vyhnout případným chybám. Počet chybně přenesených bitů ku celkovému počtu přenesených bitů se nazývá chybovost BER (Bit Error Rate). Ve snaze optimalizovat správnost přenosu jsou přenášená data ve vysílači zabezpečena vhodným ochranným kódem, čímž je původní chybovost minimalizována na tzv. chybovost zbytkovou [11].

Podstatou zabezpečovacích (ochranných) kódů je to, že přidávají k přenášeným datům redundantní informaci, díky které lze rozpoznat vzniklou chybu a případně ji i korigovat. Můžeme je tedy dělit podle zabezpečení na kódy detekční (zjišťovací) a kódy korekční (opravné) [11].

Detekční kódy jsou tedy schopné na základě redundantní informace zjistit chybnou kódovou skupinu, nicméně již nedokáží určit, ve kterém bitu skupiny chyba nastala. Detekční kódy jsou často založené na využití paritních bitů. Určitá skupina dat může být doplněna kontrolním (paritním) bitem *p*, který je např. volen tak, aby tuto skupinu doplňoval na sudý počet jedniček – tzv. sudá parita. Přijme-li přijímač kódovou skupinu bitů s lichým počtem jedniček, pak je jisté, že ve skupině nastala chyba. Chyba však může být nejen v původních informačních bitech, je též možné, že byl chybně přenesen kontrolní bit *p*. Navíc toto kódování odhalí pouze lichý počet chyb ve skupině, při sudém počtu chyb k detekci vůbec nedojde. Dokonalejšího zabezpečení lze dosáhnout vícenásobnou paritou, tedy větším počtem kontrolních bitů [11].

U korekčních kódů je parita složena z dostatečného počtu bitů vhodně vypočtených z informačních bitů tak, že je na základě chyb v paritě možné určit přesnou pozici chybného bitu a tento bit opravit (u binárních kódů prostou výměnou nuly za jedničku a obráceně). Nedochází tak ke ztrátám informace v přenášené zprávě. Více paritních bitů ale zvyšuje redundanci kódu a snižuje tak účinnost přenosu. Tuto nevýhodu lze zmenšit vyšším počtem zabezpečovaných informačních bitů v jednom slově, tedy zabezpečováním delších bloků, protože s rostoucím počtem zabezpečovaných informačních bitů klesá poměr počtu bitů potřebných pro paritu k počtu bitů ve zdrojovém slově.

Na druhou stranu se zvyšuje pravděpodobnost výskytu více chyb v zabezpečovaném bloku, který poté nebude moci být opraven. Vzhledem k tomu, že je redundantní složka vložena přímo do přenášených dat a není vyžadován tzv. zpětný kanál, označuje se tato metoda korekce jako tzv. dopředná korekce chyb FEC (Forward Error Correction). U systémů se zpětným kanálem se využívá jiné opravy chyb, tyto systémy se nazývají též systémy s automatickou žádostí o opakování přenosu ARQ (Automatic Request Repetition). Tyto systémy využívají redundantní informaci pouze k detekci vzniklé chyby, a následně odešlou zpětným kanálem žádost o opakování vysílání chybné části zprávy, dokud se nedosáhne bezchybného přenosu [11].

Podskupinou korekčních kódů jsou tzv. blokové (algebraické) kódy, mezi něž patří i Hammingův kód, kterým je kódována navigační zpráva GPS. U blokových kódů člení kodér vstupní informační tok do bloků, které zpracovává individuálně. Na základě předepsaného algoritmu přidává ke každému bloku o *k* bitech *m* kontrolních (paritních) bitů, které jsou zařazeny na začátek či konec zdrojového slova. Zdrojové slovo se tedy neprolíná se slovem kódovým, tyto kódy označujeme pak jako systematické. Z hlediska zabezpečení není mezi systematickými a nesystematickými kódy rozdíl, systematické kódy ale umožňují jednodušší dekódování a zápis. Kódové slovo je tak prodlouženo na *n* bitů. Z kolika kódových (*n*) a zdrojových (*k*) bitů se blok zabezpečený konkrétním kódem skládá, je většinou označeno uspořádanou dvojicí *(n,k)* za názvem kódu [11].

Jinou podskupinou korekčních kódů jsou tzv. konvoluční kódy. Hlavním rozdílem oproti kódům blokovým je to, že zdrojové *k-tice* nejsou kódovány nezávisle. Kodér konvolučních kódů disponuje pamětí a data jsou kódována průběžně [11].

2.1.3.2 Hammingův kód

Hammingův kód je jedním z nejčastěji používaných kódů především v oblasti telekomunikací. Byl vynalezen v roce 1950 americkým matematikem Richardem Hammingem a je možné ho realizovat binární i nebinární formou. Binární Hammingův kód je charakterizován vztahem *(n,k)* = ($2^m - 1$, $2^m - 1 - m$), kde n je počet bitů kódového slova, k je počet bitů slova zdrojového a $m = n - k$ je určité celé číslo udávající počet paritních bitů. Tento kód patří mezi lineární blokové kódy, které jsou charakterizovány vlastností, že součet dvou kódových slov je opět kódové slovo. Hammingův kód je schopen detekovat a opravit jednu chybu v kódovém slově. Chybové slovo lze poznat na základě chyby v paritě. V přijímači je parita vypočtena dle daného vzorce z přijatých informačních bitů a porovnána s paritou přijatou. Jestliže se tyto parity liší pouze v jediném bitu, nastala chyba při přenosu paritních bitů, chyba v informačním bitu by v paritě způsobila chybu mnohonásobnou. Jestliže na pozici bitů, které se shodují, dosadíme nulu, a na pozici lišících se bitů dosadíme jedničku, určuje vzniklé binární číslo pozici chyby ve slově [11][12].

Konkrétní postup výpočtu parity a následné detekce a opravy chyby si můžeme ukázat např. na Hammingově kódu (15,11), kterým chceme zabezpečit zdrojové slovo 10011010111. Kód tedy obsahuje 11 informačních bitů, které jsou zabezpečeny čtyřbitovou paritou. Vytvoříme tedy tabulku o patnácti sloupcích, sloupce s indexem mocniny o základu 2 vynecháme a zbylé vyplníme informačními bity.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P0	P1	X1	P2	X2	X3	X4	P3	X5	X6	X7	X8	X9	X10	X11
		1		0	0	1		1	0	1	0	1	1	1

Tabulka 1 - Postup výpočtu parity u Hammingova kódu (15,11)

Paritní bity jsou v tabulce označeny písmenem P a informační bity písmenem X. Rovnice pro výpočet jednotlivých paritních bitů je vždy exkluzivní součet bitů na určitých pozicích, který je roven nule. Prvním bitem tohoto součtu je zjišťovaný paritní bit a podle indexu *i* tohoto paritního bitu je

vždy 2^i bitů do rovnice zapsáno a stejný počet bitů je přeskočen. Tímto postupem vzniknou následující rovnice, kde písmenem A je označena pozice bitu v tabulce č. 1:

$$A1 \oplus A3 \oplus A5 \oplus A7 \oplus A9 \oplus A11 \oplus A13 \oplus A15 = P0 \oplus X1 \oplus X2 \oplus X4 \oplus X5 \oplus X7 \oplus X9 \oplus X11 = 0 \quad (1.a)$$

$$A2 \oplus A3 \oplus A6 \oplus A7 \oplus A10 \oplus A11 \oplus A14 \oplus A15 = P1 \oplus X1 \oplus X3 \oplus X4 \oplus X6 \oplus X7 \oplus X10 \oplus X11 = 0 \quad (1.b)$$

$$A4 \oplus A5 \oplus A6 \oplus A7 \oplus A12 \oplus A13 \oplus A14 \oplus A15 = P2 \oplus X2 \oplus X3 \oplus X4 \oplus X8 \oplus X9 \oplus X10 \oplus X11 = 0 \quad (1.c)$$

$$A8 \oplus A9 \oplus A10 \oplus A11 \oplus A12 \oplus A13 \oplus A14 \oplus A15 = P3 \oplus X5 \oplus X6 \oplus X7 \oplus X8 \oplus X9 \oplus X10 \oplus X11 = 0 \quad (1.d)$$

Úpravou těchto rovnic dojdeme k následujícímu výsledku:

P0 = 0, P1 = 1, P2 = 0, P3 = 1

Paritní bity vypočtené z uvedených rovnic zařadíme na konec zdrojového slova, čímž vznikne kódové slovo 100110101110101 připravené pro přenos. Nyní uvažujme, že je špatně přenesen pátý bit zdrojového slova, tedy bit, který je v předchozí tabulce na pozici 9. Pro lepší přehlednost si tabulku znovu přepíšeme, tentokrát již s upraveným bitem X5´ (je v tabulce č. 2 zvýrazněn).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P0´	P1´	X1´	P2´	X2´	X3´	X4´	P3´	**X5´**	X6´	X7´	X8´	X9´	X10´	X11´
		1		0	0	1		**0**	0	1	0	1	1	1

Tabulka 2 - Postup opravy u Hammingova kódu (15,11)

Opět si dle předchozích vzorců vypočteme paritní bity (tato operace bývá následně provedena v dekodéru na přijímací straně). Výsledek je následující:

P0´= 1, P1´= 1, P2´= 0, P3´= 0

V dalším kroku porovnáme paritu přijatou (vypočtenou v kodéru vysílače a přenesenou do dekodéru přijímače) a paritu vypočtenou v dekodéru z přijatých informačních bitů.

Přijatá parita	0	1	0	1
Vypočtená parita	1	1	0	0
Př. par. ⊕ vyp. par.	1	0	0	1

Tabulka 3 - Zjištění pozice chybného bitu u Hammingova kódu (15,11)

Vyjádříme-li jedničkou bity, které se u parit liší, a nulou ty, které se shodují, získáme v binární soustavě pozici chybného bitu v tabulce, v tomto případě je to kombinace bitů 1001, dekadicky tedy číslo 9.

Variantou Hammingova kódu je tzv. rozšířený Hammingův kód. Tento kód může být též znám pod názvem SECDED kód (Single Error Correction, Double Error Detection) [13]. Jak už napovídá význam této zkratky, rozšířený Hammingův kód je schopen opravit 1 chybu a 2 chyby detekovat. Nevýhodou původního (nerozšířeného) Hammingova kódu je to, že nelze rozeznat, zda ve slově nastala jedna či více chyb, tudíž i v případě vícenásobné chyby může být kódové slovo považováno za slovo s jednou chybou a může být následně špatně opraveno. Podstatou rozšířeného kódu je přidání dalšího paritního bitu za účelem rozlišení vzniku jedné a dvou chyb. Zároveň se předpokládá, že pravděpodobnost výskytu tří a více chyb v kódovém slově je zanedbatelná. Kód pak bude charakterizován uspořádanou dvojicí *(n,k)*, kde počet zdrojových bitů je zachován, tedy $k = 2^m - 1 - m$, a kde počet bitů kódového slova je zvýšen o jedna, tedy $n = 2^m$. Nově přidaný paritní bit nabývá hodnot nuly nebo jedničky tak, aby případná chyba v kódovém slově způsobila lichý počet chyb v paritních bitech.

Za předpokladu velmi nízké pravděpodobnosti vzniku tří a více chyb v kódovém slově mohou tedy nastat tyto 3 varianty [13]:

- Parita přijatá se naprosto shoduje s paritou vypočtenou z přijatých informačních (zdrojových) bitů → při přenosu chyba nenastala.
- Přijatá a vypočtená parita se neshodují v lichém počtu bitů → při přenosu nastala 1 chyba, jejíž pozici je možné určit z parity.
- Přijatá a vypočtená parita se neshodují v sudém počtu bitů → při přenosu nastaly 2 chyby, jejichž pozici nelze určit.

2.1.3.3 Konkrétní kódování navigační zprávy GPS

Pro kódování navigační zprávy GPS se používá rozšířený Hammingův kód (32,26) [10]. Z předchozích odstavců je již známo, že toto označení znamená, že se kódové slovo skládá ze 32 bitů a zdrojové (informační) slovo z 26 bitů. Zbývajících 6 bitů v kódovém slově je paritních. V podkapitole *Rozložení navigační zprávy* bylo zmíněno, že se jedno kódové slovo navigační zprávy skládá ze 30 bitů. Pro kódování Hammingovým kódem (32,26) je tedy potřeba dalších 2 bitů. Pro účely kódování jsou použity 2 bity ze slova předchozího, tj. poslední 2 paritní bity předcházejícího slova. Dekódované slovo pak obsahuje 24 zdrojových bitů. To znamená, že se dekódovaný podrámec skládá celkem z 240 bitů (z deseti dekódovaných slov).

Konkrétní vzorce pro výpočet paritních bitů v případě navigační zprávy GPS jsou uvedeny v ICD dokumentu [10] a též na následujícím obrázku č. 5:

$$
\begin{aligned}
D_1 &= d_1 \oplus D_{30}^{\star} \\
D_2 &= d_2 \oplus D_{30}^{\star} \\
D_3 &= d_3 \oplus D_{30}^{\star} \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
D_{24} &= d_{24} \oplus D_{30}^{\star} \\
D_{25} &= D_{29}^{\star} \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \oplus d_{18} \oplus d_{20} \oplus d_{23} \\
D_{26} &= D_{30}^{\star} \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{18} \oplus d_{19} \oplus d_{21} \oplus d_{24} \\
D_{27} &= D_{29}^{\star} \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{19} \oplus d_{20} \oplus d_{22} \\
D_{28} &= D_{30}^{\star} \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{20} \oplus d_{21} \oplus d_{23} \\
D_{29} &= D_{30}^{\star} \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 \oplus d_{10} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \oplus d_{21} \oplus d_{22} \oplus d_{24} \\
D_{30} &= D_{29}^{\star} \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus d_{13} \oplus d_{15} \oplus d_{19} \oplus d_{22} \oplus d_{23} \oplus d_{24}
\end{aligned}
$$

Obrázek 5 - Vzorce pro výpočet kódového slova navigační zprávy systému GPS [10]

V uvedených vzorcích jsou d_1, d_2,…, d_{24} bity zdrojového slova, D_1, D_2,…, D_{30} bity kódového slova, D_{29}* a D_{30}* poslední 2 bity předchozího kódového slova a symbolem $\oplus$ je značena operace „modulo 2“ neboli „exkluzivní součet“.

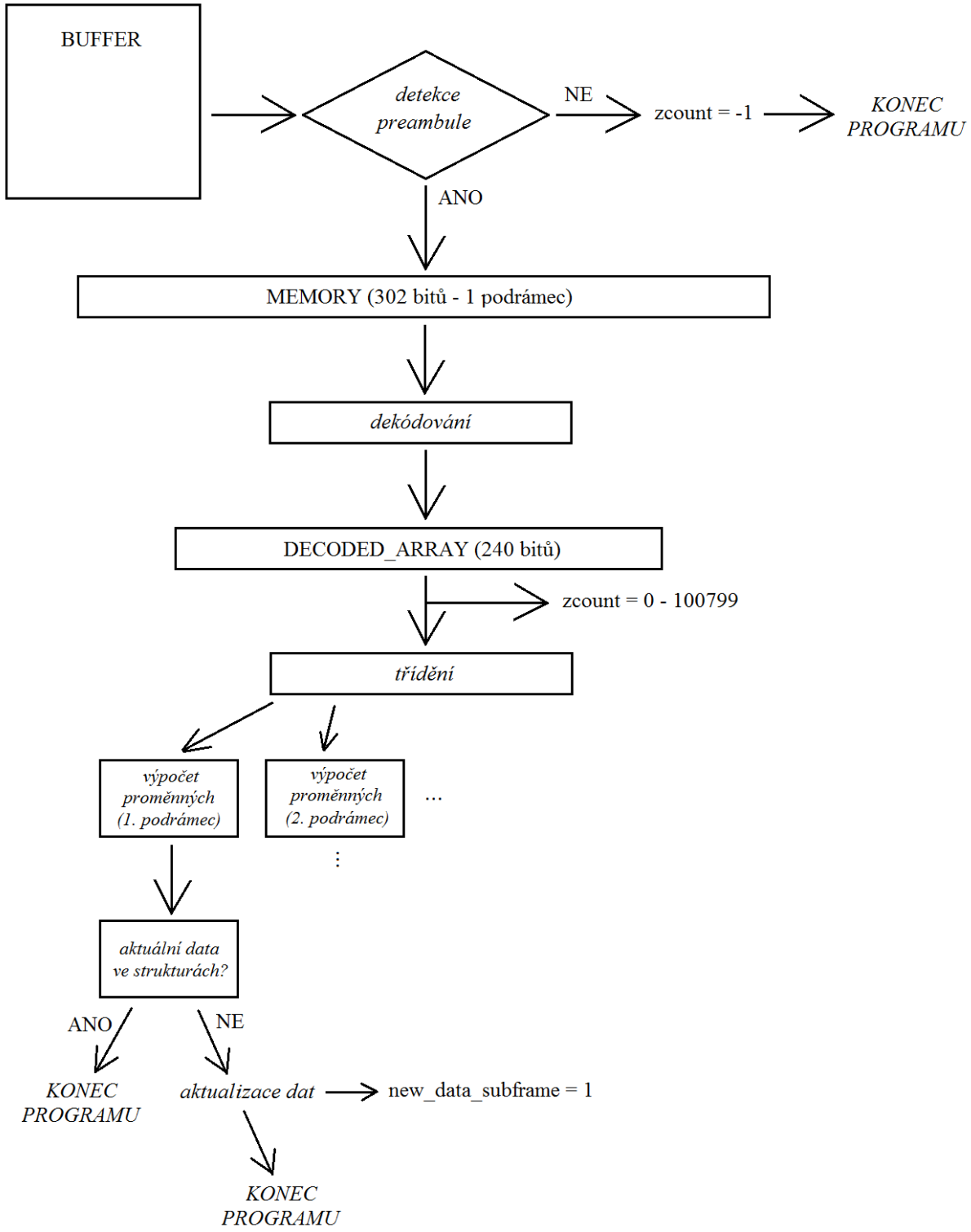
2.2 Realizace programu

2.2.1 Stavba programu

Popisovaný program je používán jako funkce, která je v hlavním programu volána vždy při přijetí nového demodulovaného bitu do bufferu. Vstupem programu je odkaz na tento buffer a číslo družice, ze které jsou data přijímána. V programu je jako první volána funkce *find_preamble*. V této funkci je zkontrolováno posledních 8 bitů bufferu a v případě nalezení preambule je zjišťováno, zda se preambule nachází i o 300 bitů dříve (tím je vyloučena většina „falešných“ preambulí – vysvětleno níže). Pokud není nalezena opakující se preambule, je do z-countu uložena hodnota -1. Vzhledem k tomu, že z-count nabývá pouze nezáporných hodnot, je tímto naznačeno, že nebyl zasynchronizován podrámec. Poté program skončí.

Pokud je nalezena opakující se preambule, je vytvořena mezipaměť MEMORY, kde je uložen poslední přijatý podrámec. Ten je následně dekódován a případně opraven funkcí *decode* a uložen do pole DECODED_ARRAY. V této fázi je vypočten z-count. Dekódovaný podrámec je následně předán do funkce *separate*, kde je určeno, jak má být dále zpracován, a na základě toho je odeslán do příslušné funkce pro zpracování proměnných. Funkcí pro zpracování proměnných je celkem 7 – jednotlivé funkce pro první, druhý a třetí podrámec, dále společná funkce pro 2., 3., 4., 5., 7., 8., 9. a 10. stránku čtvrtého podrámce a 1. až 24. stránku pátého podrámce, jednotlivé funkce pro 18. stránku čtvrtého podrámce, pro 25. stránku čtvrtého podrámce a pro 25. stránku pátého podrámce. Vypočtené hodnoty proměnných jsou porovnány s aktuálními daty a při neshodě se data aktualizují. Program je ukončen.

Náčrt programu je zobrazen na obrázku č. 6:



Obrázek 6 - Stavba programu pro zpracování navigační zprávy systému GPS

2.2.2 Synchronizace navigační zprávy

Data jsou z družice přijímána nepřetržitě. Aby bylo možné dekódovat navigační zprávu, je potřeba nejdříve nalézt začátek této zprávy, nebo alespoň začátek její dílčí části. K tomuto je určena preambule – osmibitové synchronizační slovo umístěné na začátku každého podrámce, jak bylo zmíněno výše.

Jeden zakódovaný podrámec obsahuje 10 kódových slov, každé po třiceti bitech, tj. podrámec je tvořen 300 bity [10]. To znamená, že se preambule bude periodicky opakovat každých 300 bitů. Pokud je ve zprávě nalezena preambule, která se pravidelně neopakuje, je tato preambule „falešná“, tj. byl nalezen úsek osmi bitů, který se náhodou shoduje s preambulí.

Je potřeba vzít v potaz též to, že konkrétní kódování navigační zprávy GPS bity v některých slovech invertuje. Může se proto stát, že se preambule bude nacházet v kódovém slově, jehož zdrojové bity byly invertovány a preambule samotná je tudíž také invertována. Preambule systému GPS je 10001011. Pro správnou synchronizaci je tedy nutné hledat i kombinaci bitů 01110100. Jestli budou bity v daném slově invertovány, záleží na posledním bitu parity předchozího slova D_{30}*, viz vzorce na obrázku č. 5. To znamená, že preambule v invertovaném i původním stavu budou rozmístěny zcela náhodně a je proto potřeba vyhledávat periodicky se opakující úsek, který odpovídá preambuli v jednom z těchto dvou stavů.

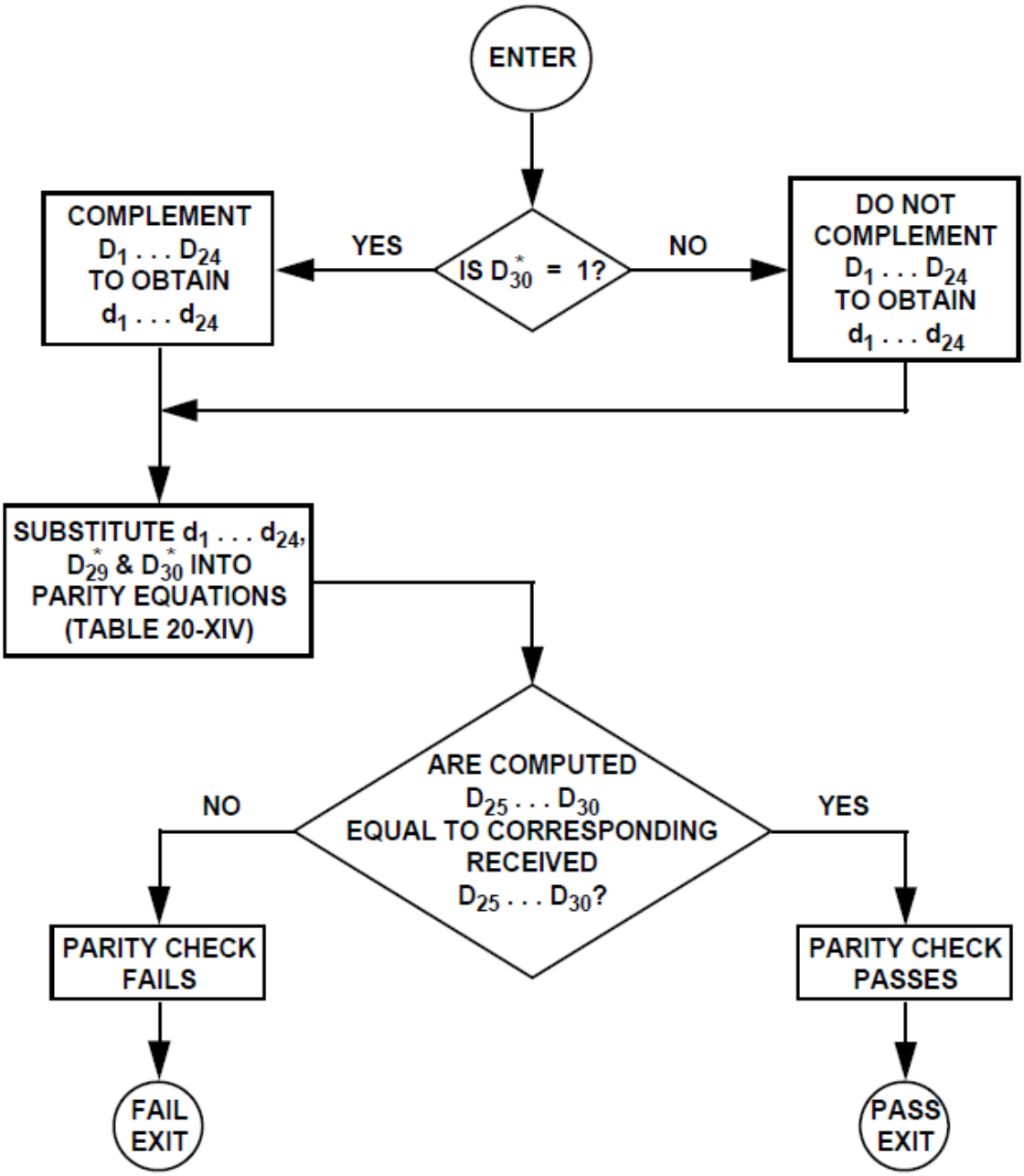
Vzhledem k tomu, že jsou v programu funkcí *find_preamble* kontrolovány pouze poslední 2 preambule, se může stát, že se v řadě bitů náhodou vyskytnou dvě „falešné“ preambule vzdálené přesně 300 bitů. Ty budou ale následně odhaleny při dekódování „falešného“ podrámce, protože s téměř stoprocentní jistotou nebude odpovídat parita. Nedojde tedy k výpočtu a uchování chybných dat.

2.2.3 Dekódování navigační zprávy

Po nalezení preambule je možné zprávu dekódovat. Pro dekódování slova je vždy potřeba si uložit i poslední 2 bity předchozího kódového slova D_{29}* a D_{30}*, viz vzorce na obrázku č. 5. K určení zdrojových bitů d_1 až d_{24} je potřeba pouze zjistit hodnotu bitu D_{30}*. Nabývá-li bit D_{30}* hodnoty 0, pak se zdrojové bity d_1 až d_{24} rovnají bitům D_1 až D_{24} kódového slova, pokud je však bit D_{30}* roven jedné, je potřeba bity D_1 až D_{24} invertovat.

Dalším krokem je ověření správnosti dekódované zdrojové informace. K tomu poslouží zbylé bity kódového slova D_{25} až D_{30} (tzv. paritní bity). Ověření správnosti je řešeno tak, že je z dekódovaných zdrojových bitů opět vypočtena parita D_{25}´až D_{30}´ dle vzorců, která je následně porovnána s přijatými bity parity D_{25} až D_{30}. Jestliže se bity přijaté parity a parity vypočtené z dekódovaných bitů shodují, byla data pravděpodobně přenesena bez chyby a proces dekódování je dokončen. V opačném případě je nutno zjistit, zda při přenosu nastala jedna či více chyb. Při detekci pouze jedné chyby je možno přijaté slovo opravit.

Na následujícím obrázku č. 7 převzatém z ICD dokumentu systému GPS [10] je zobrazen vývojový diagram, který udává doporučovaný postup při dekódování navigační zprávy. Tento postup byl v programu dodržen.



Obrázek 7 - Doporučený postup pro dekódování Hammingova kódu [10]

2.2.4 Oprava navigační zprávy

Pokud se paritní bity neshodují, nastala při přenosu zprávy chyba. Následující tabulka č. 4 vychází ze vzorců pro výpočet paritních bitů u navigační zprávy GPS. Udává, jak by paritu změnila chyba v konkrétním bitu.

Bit	Paritní bity					Součet par. b. v dek. s.	Par. bity	Součet par. b. mod. 2
	p1	p2	p3	p4	p5		p6	
d1	1	0	1	0	1	21	0	1
d2	1	1	0	1	0	26	0	1
d3	1	1	1	0	1	29	1	1
d4	0	1	1	1	0	14	0	1
d5	1	0	1	1	1	23	1	1
d6	1	1	0	1	1	27	1	1
d7	0	1	1	0	1	13	0	1
d8	0	0	1	1	0	6	1	1
d9	0	0	0	1	1	3	1	1
d10	1	0	0	0	1	17	1	1
d11	1	1	0	0	0	24	1	1
d12	1	1	1	0	0	28	0	1
d13	1	1	1	1	0	30	1	1
d14	1	1	1	1	1	31	0	1
d15	0	1	1	1	1	15	1	1
d16	0	0	1	1	1	7	0	1
d17	1	0	0	1	1	19	0	1
d18	1	1	0	0	1	25	0	1
d19	0	1	1	0	0	12	1	1
d20	1	0	1	1	0	22	0	1
d21	0	1	0	1	1	**11**	0	1
d22	0	0	1	0	1	5	1	1
d23	1	0	0	1	0	18	1	1
d24	0	1	0	0	1	9	1	1
D25	1	0	0	0	0	16	0	1
D26	0	1	0	0	0	8	0	1
D27	0	0	1	0	0	4	0	1
D28	0	0	0	1	0	2	0	1
D29	0	0	0	0	1	1	0	1
D29*	1	0	1	0	0	20	1	1
D30*	0	1	0	1	1	**11**	0	1

Tabulka 4 - Přehled chyb v paritních bitech způsobených jednou chybou v kódovém slově navigační zprávy systému GPS

Nuly v tabulce č. 4 označují bity, ve kterých se přenesená a vypočtená parita shoduje, jedničky označují bity, ve kterých se parity liší. Tento výsledek je možné získat použitím funkce bitového exkluzivního součtu – XOR (v jazyce C značeno „^“).

Tabulka potvrzuje následující tvrzení uvedená v teorii k rozšířenému Hammingovu kódu:

1. Chyba v přeneseném informačním bitu (bity d1 – d24) způsobuje v paritě vždy chybu mnohonásobnou, zatímco liší-li se parita pouze v jednom bitu, byl chybně přenesen konkrétní paritní bit.

2. Ze sedmého sloupce tabulky je zřejmé, že každý informační bit (d1 – d24) způsobí v paritě jedinečnou chybu. Ačkoli jsou vzorce pro výpočet parity u zprávy GPS tvořeny jiným postupem, než je uvedeno v teorii, lze z parity zjistit pozici chybného bitu. Chyby v paritě se shodují pouze pro bity d21 a D30* (v tabulce uvedeno tučně). Není mi zcela jasné, proč tomu tak je, v tabulce není uvedena binární kombinace 01010, dekadicky 10, která by teoreticky mohla nahradit opakující se kombinaci. Přesto tato skutečnost není velkým problémem, protože bit D30* je možné opravit již v předchozím slově. Pro program je tedy kombinace 01011, dekadicky 11, ponechána opravě bitu d21.
3. Tabulka označuje situace, kdy v kódovém slově nastane pouze jedna chyba. Z posledního sloupce tabulky je zřejmé, že počet lišících se bitů mezi přijatou a vypočtenou paritou je vždy lichý, tj. exkluzivní součet jedniček se rovná ve všech případech jedné. Na následujícím příkladu je ukázáno, že při dvou chybách v kódovém slově dojde k sudé paritě – chyby budou např. v bitech d1 a d4.

D25 = d1 ⊕ d2 ⊕ d3 ⊕ d5... → pouze jeden chybný bit → D25 bude špatně → p1 = 1

D26 = d2 ⊕ d3 ⊕ d4 ⊕ d6... → pouze jeden chybný bit → D26 bude špatně → p2 = 1

D27 = d1 ⊕ d3 ⊕ d4 ⊕ d5... → oba chybné bity → D27 bude dobře → p3 = 0

D28 = d2 ⊕ d4 ⊕ d5 ⊕ d6... → pouze jeden chybný bit → D28 bude špatně → p4 = 1

D29 = d1 ⊕ d3 ⊕ d5 ⊕ d6... → pouze jeden chybný bit → D29 bude špatně → p5 = 1

D30 = d3 ⊕ d5 ⊕ d6 ⊕ d8... → žádný chybný bit → D30 bude dobře → p6 = 0

V programu je oprava Hammingova kódu řešena v několika krocích. Nejprve se zjišťuje, zda chyba nenastala pouze v posledním bitu parity. Pokud ano, je tento bit v mezipaměti opraven, protože je zásadní pro zpracování dalšího kódového slova. Pokud ne, pak program zjišťuje, zda je v paritě sudý či lichý počet chybných bitů. V případě sudého počtu je do proměnné error_word vložena hodnota 1, což znamená, že v řešeném podrámci je minimálně jedno chybové slovo. Vzhledem k tomu, že pro další výpočty jsou potřebné všechny proměnné v podrámci a chybové slovo by způsobilo chybu v jedné nebo i více proměnných, nejde tento podrámec do dalšího zpracování, tj. je zahozen. Lichý počet chybných bitů v paritě naznačuje, že je v podrámci lichý počet chyb. Nejpravděpodobnější možností je, že nastala chyba pouze jedna, proto je slovo opraveno dle tabulky výše, která je řešena „switchem“. Pokud switch ve svém seznamu danou kombinaci nenajde, uloží program hodnotu 1 do proměnné error_word.

2.2.5 Třídění, výpočet a správa dat

Pro třídění dat je nejdříve nutné určit číslo daného podrámce. U 4. a 5. podrámce je zapotřebí určit i stránku podrámce, neboť mají stránky různá rozložení proměnných a data z nich musí být tedy poslána do různých funkcí pro výpočet těchto proměnných.

Ve funkcích pro výpočet proměnných je přesně určeno, kde se jaká proměnná v daném podrámci nachází, z kolika bitů se skládá, jakou hodnotu má nejvyšší bit a zda se počítá klasicky nebo přes binární doplněk. Všechny tyto informace lze nalézt v ICD dokumentu [10].

Správa dat je řešena tak, že program ukládá kompletní aktuální a druhá aktuální data do struktur náležících k jednotlivým podrámcům, resp. stránkám podrámců. Nově vypočtené proměnné se porovnají s hodnotami aktuálních proměnných, a pokud se neshodují, dojde k aktualizaci dat – druhá aktuální data se zahodí a jsou nahrazena původními aktuálními daty a na místo aktuálních dat jsou uložena data právě vypočtená.

Na správu dat byly zatím kladeny 2 požadavky. Za prvé byla požadována hláška, díky které by se poznalo, že data byla právě aktualizována. Tento problém je řešen globální proměnnou new_data_subframe, která nabývá při každém běhu programu hodnot 1-7 podle toho, který podrámec, resp. stránka podrámce byla aktualizována. Druhým požadavkem byla detekce chybového slova v podrámci. Proto byla do programu přidána proměnná error_word, která byla zmíněna výše. Nabývá-li tato proměnná hodnoty 1, je podrámec automaticky zahozen, protože chybějící informace by v následném zpracování mohly ovlivnit hodnoty jedné i více proměnných a následně by byla chybně spočtena poloha družice.

3 Navigační systém BeiDou

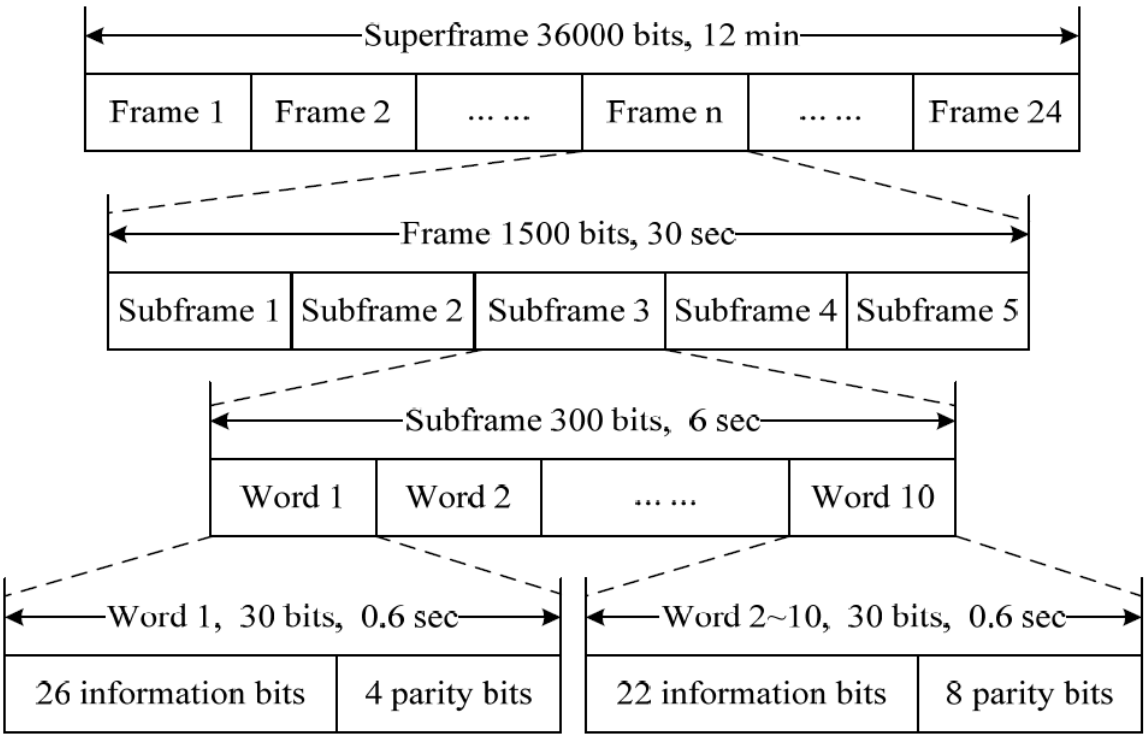
3.1 Navigační zpráva systému BeiDou

3.1.1 Rozložení a obsah navigační zprávy systému BeiDou

Navigační zpráva systému BeiDou je na rozdíl od zprávy GPS rozdělena do dvou formátů obecně nazvaných D1 a D2. Tyto formáty se od sebe liší především strukturou obsažených dat a přenosovou rychlostí. Navigační zpráva formátu D1 je vysílána družicemi MEO a IGSO přenosovou rychlostí 50 b/s a je modulována sekundárním kódem o rychlosti 1 kb/s. Tento formát obsahuje základní navigační informace – efemeridy vysílající družice a almanachy ostatních družic, a je vlastně velmi podobný navigační zprávě systému GPS. Navigační zpráva formátu D2 je vysílána geostacionárními družicemi desetinásobnou rychlostí vůči formátu D1, tedy přenosovou rychlostí 500 b/s. Kromě základních informací obsahuje tento formát také přesnější servisní informace, např. informace o integritě systému a rozsáhlejší informace o stavu ionosféry [7].

Navigační zpráva je opět dělena do rámců, podrámců a jednotlivých slov. Zpráva formátu D1 obsahuje 24 rámců, zpráva formátu D2 se dělí na 120 rámců. Dále už je dělení u obou formátů stejné. V každém rámci je obsaženo 5 podrámců, každý z nich je tvořen 300 bity. Podrámce jsou dále děleny na deset třicetibitových slov, která jsou kódována zvlášť [7].

První, druhý a třetí podrámec formátu D1 obsahuje efemeridy, čtvrtý a pátý podrámec pak almanachy a další data, která slouží např. ke spolupráci s ostatními systémy. Zatímco u formátu D1 nebyly u prvních tří podrámců rozlišovány jednotlivé stránky, u formátu D2 je tomu jinak. První podrámec formátu D2 je dělen do deseti stránek a obsahuje celé efemeridy. Druhý a třetí podrámec se skládají ze 6 stránek a obsahují nové proměnné, které nebyly ve formátu D1 uvedeny. Ze 6 stránek je tvořen i čtvrtý podrámec, který obsahuje pouze rezervované bity. Na 120 stránek se dělí jen pátý podrámec obsahující především almanachy a rozsáhlá ionosférická data. Více o této problematice lze nalézt v [7].



Obrázek 8 - Struktura formátu D1 navigační zprávy systému BeiDou [7]

3.1.2 Kódování navigační zprávy systému BeiDou

3.1.2.1 BCH kódy

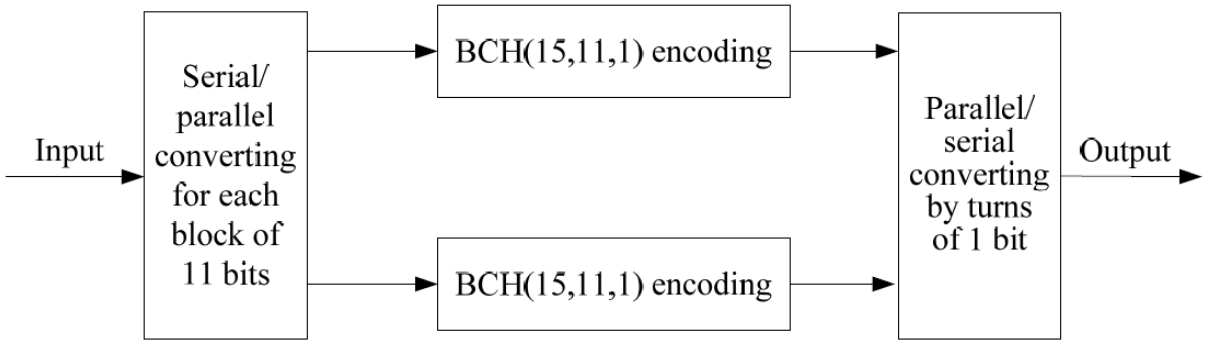
BCH kódy patří do skupiny tzv. cyklických kódů, která je důležitou podtřídou lineárních blokových kódů. Jak již název napovídá, základní vlastností cyklických kódů je to, že cyklickým posuvem určitého kódového slova vzniká opět kódové slovo. Další podstatnou vlastností těchto kódů je schopnost zabezpečení proti shlukům chyb, jejichž délka je rovna nebo je menší než počet paritních bitů, tj. počet paritních bitů je volen na základě požadovaného stupně zabezpečení. Cyklické kódy jsou hojně využívány také díky jejich relativně snadné implementaci např. pomocí posuvných registrů.

BCH kódy (Bose, Chaudhuri, Hocquencgham Codes) byly vynalezeny na přelomu 50. a 60. let minulého století. Základní variantou BCH kódů jsou kódy binární, jejich zevšeobecněním lze však vytvořit i nebinární kódy. BCH kódy jsou charakterizovány třemi parametry *(n,k,t)*, kde $n = 2^m - 1$; $m \geq 3$ je počet bitů kódového slova, k je počet bitů informačního (zdrojového) slova a $t < (2^m - 1)/2$ je počet opravitelných chyb [11][12].

3.1.2.2 Prokládání

Účinnost ochrany dat při přenosu je mimo jiné závislá na tom, jak jsou chyby vzniklé při přenosu v datech rozloženy. Podle charakteru rozložení mohou být chyby rozděleny na ojedinělé (nezávislé) chyby a shluky chyb (tzv. bursty). Výskyt ojedinělých chyb v jednotlivých kódových blocích je statisticky nezávislý. Vhodně zvoleným ochranným kódem je možné tyto chyby ve většině případů detekovat a opravit. Shluk chyb je charakterizován intervalem, ve kterém se chybovost oproti úseku s ojedinělými chybami mnohonásobně zvýší. Objeví-li se tento shluk v kódovém bloku, pak ho ochranný kód pravděpodobně detekuje, ale není už schopen ho opravit [11].

Prokládání (interleaving) je jedna z nejčastějších metod pro korekci shluku chyb. Princip prokládání je v tom, že se přenášené bity ve vysílači vkládají do paměťové matice po řádcích, a následně jsou z této matice čteny po sloupcích a přenášeny komunikačním kanálem. V přijímači se původní bitová sekvence získá pomocí stejné paměťové matice s tím rozdílem, že jsou přijaté bity do matice vkládány po sloupcích a z ní čteny po řádcích. Tímto postupem je případný shluk chyb v bitové sekvenci rozptýlen, tj. přeměněn na ojedinělé chyby, které lze snadněji korigovat [11].



Obrázek 9 – Kódování a prokládání navigační zprávy systému BeiDou [7]

3.1.2.3 Konkrétní kódování navigační zprávy

Zpráva je kódována BCH kódem (15,11,1) [7]. Kódové slovo kódované tímto BCH kódem tedy obsahuje 15 bitů, z nichž 11 bitů je zdrojových a 4 bity jsou paritní, a daný BCH kód je schopný v tomto slově detekovat a odstranit jednu chybu. Dle zdroje [14] se BCH kód o délce kódového slova $2^m - 1$ a se schopností opravit pouze jednoduchou chybu chová jako Hammingův kód, což ukazuje tabulka syndromů z ICD dokumentu [7] (obrázek č. 10).

$D_3D_2D_1D_0$	15 bits data for error correction
0000	000000000000000
0001	000000000000001
0010	000000000000010
0011	000000000010000
0100	000000000000100
0101	000000100000000
0110	000000000100000
0111	000010000000000
1000	000000000001000
1001	100000000000000
1010	000001000000000
1011	000000010000000
1100	000000001000000
1101	010000000000000
1110	000100000000000
1111	001000000000000

Obrázek 10 - Tabulka syndromů BCH kódu (15,11,1) použitého při kódování navigační zprávy systému BeiDou [7]

Dříve již bylo zmíněno, že se podrámec navigační zprávy BeiDou skládá z deseti třicetibitových kódových slov. První kódové slovo se od zbylých kódových slov liší. Obsahuje 26 informačních bitů, z nichž prvních 15 bitů není kódováno a zbylých 11 tvoří blok zabezpečený uvedeným BCH kódem. Struktura prvního slova je naznačena v tabulce č. 5:

X_1^1	X_1^2	X_1^3	...	X_1^{14}	X_1^{15}	X_2^1	X_2^2	X_2^3	...	X_2^{11}	P_2^1	P_2^2	P_2^3	P_2^4

Tabulka 5 - Struktura prvního kódového slova podrámce v navigační zprávě systému BeiDou

Písmenem X jsou označeny informační bity, písmenem P paritní bity. Dolní index určuje, ze kterého kódového bloku daný bit pochází, tedy nabývá hodnot 1 nebo 2. Horní index značí pořadí bitu v daném bloku, u informačních bitů obvykle nabývá hodnot 1 až 11, u paritních bitů 1 až 4 s tím, že se všechny paritní bity řadí vždy až za bity informační. Výjimkou je již zmíněná první polovina prvního kódového slova podrámce, která není kódována, a tudíž netvoří paritní bity a horní index u informačního bitu nabývá až hodnoty 15.

Druhé až desáté slovo podrámce obsahuje 22 informačních a 8 paritních bitů. Informační bity jsou rozděleny do dvou bloků a zvlášť kódovány. Třicetibitové slovo pak vzniká prokládáním těchto dvou kódovaných bloků. Rozložení slov je zobrazeno na obrázku č. 6, význam značení je uveden v předchozím odstavci [7].

X_1^1	X_2^1	X_1^2	X_2^2	…	X_1^{11}	X_2^{11}	P_1^1	P_2^1	P_1^2	P_2^2	P_1^3	P_2^3	P_1^4	P_2^4

Tabulka 6 - Struktura druhého až desátého kódového slova podrámce v navigační zprávě systému BeiDou

3.2 Realizace programu

Program pro zpracování zprávy navigačního systému BeiDou bude velmi podobný programu realizovanému pro systém GPS. Vstupem programu bude kromě odkazu na buffer s demodulovanými bity a čísla družice, ze které je signál přijímán, také informace o tom, který formát je přijímán (zda formát D1 nebo D2). Výstupem bude parametr SOW, který nabývá nezáporných hodnot v případě, že byl nalezen začátek podrámce, nebo hodnoty -1, pokud podrámec zasynchronizován nebyl.

3.2.1 Synchronizace navigační zprávy

K synchronizaci navigační zprávy opět slouží tzv. preambule. V případě systému BeiDou je to jedenáctibitové slovo opakující se po 300 bitech. Preambule se nachází v nezakódované části prvního slova každého podrámce [7]. To vedlo k původní myšlence, že je třeba vyhledávat pouze preambuli v původním tvaru, tj. bitovou posloupnost 11100010010. Na základě testování ale bylo zjištěno, že se mohou bity navigační zprávy (a tudíž i preambule) vyskytovat i v inverzní podobě. Inverze bitů je pravděpodobně způsobena při demodulování BPSK modulace. BPSK modulace je schopna rozlišit 2 stavy, ale nedokáže určit, který ze stavů nabýval ve vysílači hodnoty jedna a který hodnoty nula. Může se tak tedy stát, že jsou stavům přiřazeny hodnoty obráceně, než jak tomu bylo v původní bitové sekvenci.

Invertovaná preambule je jedinou možností, jak zjistit, že jsou invertovány i další bity navigační zprávy. Je proto potřeba si uchovat informaci o tom, zda byla preambule v původní či invertované podobě. Tato informace bude pak jedním ze vstupů funkce pro dekódování podrámce, na jejímž základě se bude rozhodovat, zda je nutné invertovat následující bity.

3.2.2 Dekódování a oprava navigační zprávy

Následující tabulka vznikla úpravou tabulky syndromů (obrázek č. 10).

Bit	Paritní bity				Součet par. b. v dek. s.
	D3	D2	D1	D0	
X1	1	0	0	1	9
X2	1	1	0	1	13
X3	1	1	1	1	15
X4	1	1	1	0	14
X5	0	1	1	1	7
X6	1	0	1	0	10
X7	0	1	0	1	5
X8	1	0	1	1	11
X9	1	1	0	0	12
X10	0	1	1	0	6
X11	0	0	1	1	3
P1	1	0	0	0	8
P2	0	1	0	0	4
P3	0	0	1	0	2
P4	0	0	0	1	1

Tabulka 7 - Přehled chyb v paritních bitech způsobených jednou chybou v kódovém slově navigační zprávy systému BeiDou

Tabulka opět ukazuje, jakou chybu v paritních bitech by způsobila jedna chyba v konkrétním bitu kódového slova. Vzorce pro výpočet paritních bitů nejsou v ICD dokumentu uvedeny, nicméně se dají z uvedené tabulky snadno odvodit. Pro výpočet určitého paritního bitu D3 – D0 postačí pomocí exkluzivního součtu sečíst bity, které nabývají ve sloupci náležejícím paritnímu bitu při chybné paritě hodnoty jedna. Tím je zaručeno, že nastane-li chyba v jednom bitu kódového slova, bude způsobena chyba v paritních bitech přesně podle tabulky. Odvozené vzorce jsou následující:

$$D3 = X1 \oplus X2 \oplus X3 \oplus X4 \oplus X6 \oplus X8 \oplus X9 \quad (2.a)$$

$$D2 = X2 \oplus X3 \oplus X4 \oplus X5 \oplus X7 \oplus X9 \oplus X10 \quad (2.b)$$

$$D1 = X3 \oplus X4 \oplus X5 \oplus X6 \oplus X8 \oplus X10 \oplus X11 \quad (2.c)$$

$$D0 = X1 \oplus X2 \oplus X3 \oplus X5 \oplus X7 \oplus X8 \oplus X11 \quad (2.d)$$

V prvním kroku dekódování navigační zprávy BeiDou je potřeba zjistit, zda je preambule v původní či invertované verzi. Pokud je preambule invertovaná, pak je pravděpodobné, že jsou invertovány všechny bity podrámce. V tom případě je nutné bity znovu invertovat, aby nabývaly původních hodnot (tedy zaměnit nuly za jedničky a obráceně). Není-li invertován celý podrámec, nastane chyba ve slově, které je částečně původní a částečně invertované. Pak je otázkou, vyplatí-li se zpracovávat takový podrámec, jehož preambule se neshoduje s preambulí dalšího podrámce. S velkou pravděpodobností bude ve zmíněném slově chyba vícenásobná, která bude považována za jednoduchou a slovo bude tak špatně opraveno. Podrámce s minimálně jedním chybovým slovem se zahazují, protože by toto chybové slovo mohlo způsobit chybu v jedné či více proměnných, a pro další zpracování je potřeba všech správně dekódovaných proměnných v podrámci.

Po kontrole preambule a případné inverzi bitů jsou z přijatých informačních bitů každého slova vypočteny paritní bity, které jsou porovnány s přijatou paritou. Pokud se paritní bity liší, dojde k opravě slova pomocí předchozí tabulky, která bude také řešena „switchem“. Kódování zprávy

BeiDou není schopno rozlišit jednoduchou chybu v kódovém slově od mnohonásobné, proto není možné zjistit, zda bylo toto slovo správně opraveno.

3.2.3 Třídění, výpočet a správa dat

Vzhledem k tomu, že existují 2 různé formáty navigační zprávy BeiDou, je potřeba vytvořit mnohem více struktur pro uchování dat a též mnohem více funkcí pro výpočet a aktualizaci proměnných. Formát zprávy je nutné řešit až ve funkci *separate*, synchronizace a dekódování obou formátů je stejné. Pro určení vhodného dalšího zpracování je tedy potřeba zjistit formát navigační zprávy, číslo podrámce a případně i stránku podrámce.

Zpracování proměnných u formátu D1 bude provedeno velmi podobně jako u navigační zprávy systému GPS. Nový problém ale nastal u formátu D2, u kterého jsou některé proměnné rozděleny do 2 podrámců. Protože je program pro zpracování navigační zprávy pouze funkcí v hlavním programu, je potřeba v hlavním programu vytvořit globální proměnnou, do níž budou postupně uloženy obě části proměnné, a tato proměnná bude následně vypočtena. Není možné provést částečný výpočet v každém z podrámců zvlášť, protože se tyto proměnné většinou počítají přes dvojkový doplněk, u kterého je potřeba před výpočtem provést jisté úpravy bitové posloupnosti dané proměnné.

4 Závěr

Program pro zpracování navigační zprávy systému GPS byl v minulosti několikrát testován a jevil se jako funkční. Na základě nových poznatků o Hammingově kódu byla však provedena mírná úprava funkce *decode*, konkrétně její části pro opravu chybového slova. Novou verzi nebylo již možné znovu otestovat kvůli technickým problémům s prototypem multikonstelačního přijímače. Nepředpokládám ale, že by v nové verzi byly výrazné chyby. Již při programování původní verze jsem si vytvořila vlastní bitovou sekvenci, která obsahuje základní parametry reálné navigační zprávy. Při použití novější verze programu na tu samou bitovou sekvenci dochází ke stejným výsledkům jako v případě verze původní, která byla na reálných datech testována. Zároveň jsem záměrnou tvorbou chyb v bitové sekvenci ověřovala, zda oprava chybového slova funguje opravdu tak, jak je uvedeno výše.

Dříve bylo především testováno, zda program správně dekóduje Hammingův kód a zda dochází ke správnému výpočtu jednotlivých proměnných a k jejich aktualizaci. Vzhledem k tomu, že zatím nebyl vytvořen algoritmus pro následné zpracování získaných proměnných, mohou být později při následném používání navrženy ještě drobné úpravy pro vylepšení programu. Jednou z těchto úprav by mohlo být např. ukládání částečných dat při zpracování podrámce, který obsahuje chybové slovo. Pokud by tento podrámec v dalším rámci opět obsahoval chybové slovo, které by ale bylo na jiné pozici v tomto podrámci, pak by bylo teoreticky možné částečná data z těchto podrámců sloučit a dále zpracovat. Současná verze programu by oba chybové podrámce zahodila.

Realizace programu pro zpracování navigační zprávy BeiDou se též blíží ke konci. Program nemohl být dlouho testován kvůli problémům s generátorem signálu. Uvedená teorie je ověřena na základě výpočtů ze zkušebních dat. Těmito výpočty byla ověřena především teorie o dekódování signálu, tj. byla testována správnost vzorců pro výpočet paritních bitů. Zbylé části programu jsou řešeny velmi podobně jako u programu pro systém GPS, neměl by v nich tedy nastat problém.

5 Použitá literatura

[1] *Globální polohovací a navigační satelitní systémy* [online]. Dostupné z: http://geologie.vsb.cz/geoinformatika/kap09.htm

[2] Hrdina, Z.; Pánek, P.; Vejražka, F.: *Rádiové určování polohy (Družicový systém GPS)* [skriptum]. Vydavatelství ČVUT, Praha, 1995.

[3] Vejražka, F.: *Novinky v družicových navigačních systémech*. In: Radiokomunikace 2013, str. 173 – 188. UNIT, Pardubice, 2013.

[4] *Current and Future Satellite Generations* [online]. Dostupné z: http://www.gps.gov/systems/gps/space/

[5] *Global Navigation satellite System GLONASS. Interface Control Document. Navigational signal in bands L1, L2. Edition 5.1*. Russian Institute of Space Engineering, Moscow, 2008.

[6] *GALILEO – Evropský globální navigační družicový systém* [online]. Dostupné z: http://www.czechspaceportal.cz/3-sekce/gnss-systemy/galileo/

[7] *BeiDou Navigation Satellite System. Signal in Space. Interface Control Document. Open Service Signal (Version 2.1)*. China Satellite Navigation Office, November 2016.

[8] *Indian Regional Navigation Satellite System (IRNSS): NavIC* [online]. Dostupné z: http://www.isro.gov.in/irnss-programme

[9] *Overview of the Quasi-Zenith Satellite System (QZSS)* [online]. Dostupné z: http://qzss.go.jp/en/overview/services/sv01_what.html

[10] *Interface Specification IS-GPS-200G. Navstar GPS Space Segment/Navigation User Interfaces*. Global Positioning Systems Directorate, Washington, 2013.

[11] Žalud, V.: *Moderní radioelektronika*. BEN – technická literatura, Praha, 2000.

[12] Berrou, C.: *Codes and Turbo Codes*. Springer-Verlag France, Paris, 2010. ISBN 978-2-8178-0039-4

[13] Heuring, V.; Jordan, H.: *Computer Systems Design and Architecture*. Prentice Hall, 2004. Prezentace Power Point. Dostupné z: http://www.cs.wustl.edu/~jbf/cse362.d/cse362.slides.d/Ch8.pdf

[14] Han, Y. S.: *BCH Codes* [online]. National Taipei University, Graduate Institute of Communication Engineering, Taiwan, 2010. Dostupné z: http://web.ntpu.edu.tw/~yshan/BCH_code.pdf

Seznam příloh

Příloha A

```
#ifndef PROCESS_GPS_DATA_H_
#define PROCESS_GPS_DATA_H_

/* #include <stdio.h> */
#include <stdlib.h>
#include <stddef.h>
#include <inttypes.h>

/* definice struktur - jednotky promennych uvedeny v ICD dokumentu */

struct subframe1{ /* pri aktualizaci new_data_subframe = 1 */
uint16_t WN;
uint8_t kod_L2;
uint8_t URA_index;
uint8_t SV_health;
uint16_t IODC;
uint8_t L2;
double TGD;
uint32_t toc;
double af2;
double af1;
double af0;
};

struct subframe2{ /* pri aktualizaci new_data_subframe = 2 */
uint8_t IODE;
double Crs;
double delta_n;
double M0;
double Cuc;
double e;
double Cus;
double odm_A;
uint32_t toe;
uint8_t fit_interval_flag;
int AODO;
};

struct subframe3{ /* pri aktualizaci new_data_subframe = 3 */
double Cic;
double OMG0;
double Cis;
double i0;
double Crc;
double omg;
double OMGd;
```

```
uint8_t IODE;
double idot;
};

struct subframe4and5{ /* pri aktualizaci new_data_subframe = 4 */
uint8_t data_ID;
uint8_t SV_ID;
double e;
uint32_t toa;
double delta_i;
double OMGd;
uint8_t SV_health;
double odm_A;
double OMG0;
double omg;
double M0;
double af0;
double af1;
};

struct subframe4page18{ /* pri aktualizaci new_data_subframe = 5 */
uint8_t data_ID;
uint8_t SV_ID;
double alfa0;
double alfa1;
double alfa2;
double alfa3;
double beta0;
double beta1;
double beta2;
double beta3;
double A1;
double A0;
uint32_t tot;
uint8_t WNt;
int delta_tLS;
uint8_t WNLSF;
uint8_t DN;
int delta_tLSF;
};

struct subframe4page25{ /* pri aktualizaci new_data_subframe = 6 */
uint8_t data_ID;
uint8_t SV_ID;
uint8_t AS_config[32]; /*1.-32. druzice*/
uint8_t SV_health[8]; /*25.-32. druzice*/
};
```

```
struct subframe5page25{ /* pri aktualizaci new_data_subframe = 7 */
uint8_t data_ID;
uint8_t SV_ID;
uint32_t toa;
uint8_t WNa;
uint8_t SV_health[24]; /*1.-24. druzice*/
};

int32_t process_GPS_data(uint64_t * buffer, uint8_t SV_number);

#endif
```

Příloha B

```
/* #include <stdio.h> */
#include <stdlib.h>
#include <stddef.h>
#include <inttypes.h>
#include "xil_printf.h"
#include "process_GPS_data.h"

#define TEST_data /* Zakomentovanim vypnuty vypisy */

/* lokalni promenne */
uint8_t decoded_array[30]; /* ulozeni dekodovaneho podramce */

/* globalni promenne */
uint8_t new_data_subframe; /* promennou znacena aktualizace struktur */

/* efemeridy - data z jednotlivych druzic */
extern struct subframe1 sub1[30]; /* pracovní struktury, data se porovnávají s aktuálními  */
extern struct subframe1 sub1_act[30]; /* aktuální data */
extern struct subframe1 sub1_2act[30]; /* druhá aktuální data */

extern struct subframe2 sub2[30];
extern struct subframe2 sub2_act[30];
extern struct subframe2 sub2_2act[30];

extern struct subframe3 sub3[30];
extern struct subframe3 sub3_act[30];
extern struct subframe3 sub3_2act[30];

/* almanachy - spolecna data ze vsech druzic */
extern struct subframe4and5 sub4and5;
extern struct subframe4and5 sub4and5_act;
extern struct subframe4and5 sub4and5_2act;

extern struct subframe4page18 sub4p18;
```

```
extern struct subframe4page18 sub4p18_act;
extern struct subframe4page18 sub4p18_2act;

extern struct subframe4page25 sub4p25;
extern struct subframe4page25 sub4p25_act;
extern struct subframe4page25 sub4p25_2act;

extern struct subframe5page25 sub5p25;
extern struct subframe5page25 sub5p25_act;
extern struct subframe5page25 sub5p25_2act;

/* vypocet promennych klasicky - bez znamenka */
double calculate(uint8_t position, uint8_t bit_number, int MSB){ /* par: prvni bit promenne (pole
zacina 0), pocet bitu promenne, exponent u nejvyssiho bitu */
double total_result = 0;
double result = 0;
uint8_t i;
uint8_t index = (position/8);
uint8_t mask = 0x80;
mask >>= (position%8);
int exponent;
for(i = 0; i < bit_number; i++){
   if((((position+i)%8) == 0) && (i != 0)){
      mask = 0x80;
      index++;
   }
   exponent = MSB - i;

   if(exponent == 0){ /* exponent roven nule */
      if((decoded_array[index]&mask) == 0){
         result = 0;
      }else{
         result = 1;
      }
   }
   else if(exponent > 0){ /* kladny exponent */
      if((decoded_array[index]&mask) == 0){
         result = 0;
      }else{
         result = 2;
         while(exponent > 1){
            result = result * 2;
            exponent--;
      }
   }
   }
   else{ /* zaporny exponent */
      if((decoded_array[index]&mask) == 0){
```

```
        result = 0;
      }else{
        exponent = exponent*(-1);
        result = 2;
        while(exponent > 1){
          result = result * 2;
          exponent--;
        }
        result = 1/result;
   }
   }
   total_result = total_result + result;
   mask >>= 1;
}
return total_result;
}

/* vypocet promennych pres dvojkovy doplnek - se znamenkem */
double calculate_binary(uint8_t position, uint8_t bit_number, int MSB){ /* par: prvni bit promenne
(pole zacina 0), pocet bitu promenne, exponent u nejvyssiho bitu */
double total_result = 0;
uint8_t i;
uint8_t index = (position/8);
uint8_t mask = 0x80;
mask >>= (position%8);

if((decoded_array[index]&mask) == 0){ /* prvni bit promenne je nulovy (cislo je kladne) -> klasicke
reseni */
   total_result = calculate(position, bit_number, MSB);
}else{ /* prvni bit pole je nenulovy (cislo je zaporne) -> binarni doplnek */
   index = ((position+bit_number-1)/8);
   mask = 0x80;
   mask >>= ((position+bit_number-1)%8);
   while((decoded_array[index]&mask) == 0){ /* odecteni jednicky od daneho binarniho cisla */
      decoded_array[index] = (decoded_array[index]|mask);
      mask <<= 1;
      if(mask == 0){
        mask = 0x01;
        index--;
      }
   }
   decoded_array[index] = (decoded_array[index]^mask);

   index = (position/8);
   mask = 0x80;
   mask >>= (position%8);
   for(i = 0; i < bit_number; i++){ /* prepis pole na inverzni */
      decoded_array[index] = (decoded_array[index]^mask);
```

```
      mask >>= 1;
      if(mask == 0){
         mask = 0x80;
         index++;
      }
   }
   total_result = calculate(position, bit_number, MSB); /* prepocet binarniho cisla na dekadicke */
   total_result = total_result*(-1); /* zaporne znamenko */
}
return total_result;
}

/* hledani preambule */
int find_preamble(uint64_t * buffer){ /* par: ukazatel na buffer predany do fce process_GPS_data */
int is_pream = 0;
uint8_t preamble_normal = 0x8b; /* normalni preambule, binarne 10001011 */
uint8_t preamble_invers = 0x74; /* inverzni preambule, binarne 01110100 */
uint8_t preamble1;
uint8_t preamble2;
uint8_t mask = 0xff;
uint64_t buffer4 = buffer[4];
preamble1 = mask&buffer[0];
buffer4>>=44;
preamble2 = mask&buffer4;

if(preamble1 == preamble_normal){ /* hledani posledni preambule v bufferu (p. v normalni forme) */
   if(preamble2 == preamble_normal) /* hledani predposledni preambule v bufferu (p. v normalni
forme) */
      is_pream = 1;
   if(preamble2 == preamble_invers) /* hledani predposledni preambule v bufferu (p. v inverzni
forme) */
      is_pream = 1;
}
if(preamble1 == preamble_invers){ /* hledani posledni preambule v bufferu (p. v inverzni forme) */
   if(preamble2 == preamble_normal) /* hledani predposledni preambule v bufferu (p. v normalni
forme) */
      is_pream = 1;
   if(preamble2 == preamble_invers) /* hledani predposledni preambule v bufferu (p. v inverzni
forme) */
      is_pream = 1;
}

return is_pream; /* is_pream = 0 -> nezasynchronizovan podramec, is_pream = 1 -> pravdepodobne
zasynchronizovan podramec */
}

/* dekodovani a pripadna oprava jednoho zasynchronizovaneho podramce */
uint8_t decode(uint64_t * coded_array){ /* par: odkaz na kodovany podramec */
```

```
uint8_t parity_received;
uint8_t parity_calculated;
uint64_t bits, y64;
uint64_t mask64;
uint32_t coded_word;
uint32_t mask32 = 0xffffffff;
uint8_t mask8;
uint8_t i,j,x,y,check;
uint8_t position_array;
uint8_t position_bit;
uint8_t error_word = 0;

for(i = 0; i < 10; i++){ /* kopirovani jednoho kodoveho slova do promenne coded_word */
   coded_word = 0;
   mask64 = 0x00000000ffffffff;
   if(i == 0){ /* prvni slovo */
      bits = coded_array[0];
      bits >>= 32;
      coded_word = (bits&mask64);
   }else if((i%2) == 1){ /* suda slova */
      bits = coded_array[(30*i/64)];
      bits >>= (2*i);
      coded_word = (bits&mask64);
   }else{ /* licha slova */
      bits = coded_array[(30*i/64)];
      bits <<= (32-2*i);
      coded_word = (bits&mask64);
      bits = coded_array[(30*i/64)+1];
      bits >>= (32+2*i);
      coded_word = (coded_word|bits);
   }
   mask32 = 0x0000003f;
   parity_received = (coded_word&mask32); /* ulozeni prijate parity */
   mask32 = 0x40000000;
   mask8 = 0xff;
   if((coded_word&mask32) == 0){ /* dekodovani slova pro pripad D30* = 0 */
      coded_word >>= 6;
      for(j = 3; j > 0; j--){
         decoded_array[j-1+(i*3)] = coded_word&mask8;
         coded_word >>= 8;
      }
   }else{ /* dekodovani slova pro pripad D30* = 1 */
      coded_word >>= 6;
      for(j = 3; j > 0; j--){
         decoded_array[j-1+(i*3)] = ~(coded_word&mask8);
         coded_word >>= 8;
      }
   }
```

```
   /* vypocet parity z dekodovaneho slova */
   parity_calculated = 0;

j=(((decoded_array[i*3]&0x80)>>7)+((decoded_array[i*3]&0x40)>>6)+((decoded_array[i*3]&0x20)>>
5)+((decoded_array[i*3]&0x08)>>3)+((decoded_array[i*3]&0x04)>>2)+((decoded_array[i*3+1]&0x40
)>>6)+((decoded_array[i*3+1]&0x20)>>5)+((decoded_array[i*3+1]&0x10)>>4)+((decoded_array[i*3+
1]&0x08)>>3)+((decoded_array[i*3+1]&0x04)>>2)+((decoded_array[i*3+2]&0x80)>>7)+((decoded_ar
ray[i*3+2]&0x40)>>6)+((decoded_array[i*3+2]&0x10)>>4)+((decoded_array[i*3+2]&0x02)>>1)+((cod
ed_word&0x02)>>1))%2;
   j <<= 5;
   parity_calculated = (parity_calculated|j);

j=(((decoded_array[i*3]&0x40)>>6)+((decoded_array[i*3]&0x20)>>5)+((decoded_array[i*3]&0x10)>>
4)+((decoded_array[i*3]&0x04)>>2)+((decoded_array[i*3]&0x02)>>1)+((decoded_array[i*3+1]&0x20
)>>5)+((decoded_array[i*3+1]&0x10)>>4)+((decoded_array[i*3+1]&0x08)>>3)+((decoded_array[i*3+
1]&0x04)>>2)+((decoded_array[i*3+1]&0x02)>>1)+((decoded_array[i*3+2]&0x40)>>6)+((decoded_ar
ray[i*3+2]&0x20)>>5)+((decoded_array[i*3+2]&0x08)>>3)+(decoded_array[i*3+2]&0x01)+(coded_w
ord&0x01))%2;
   j <<= 4;
   parity_calculated = (parity_calculated|j);

j=(((decoded_array[i*3]&0x80)>>7)+((decoded_array[i*3]&0x20)>>5)+((decoded_array[i*3]&0x10)>>
4)+((decoded_array[i*3]&0x08)>>3)+((decoded_array[i*3]&0x02)>>1)+(decoded_array[i*3]&0x01)+(
(decoded_array[i*3+1]&0x10)>>4)+((decoded_array[i*3+1]&0x08)>>3)+((decoded_array[i*3+1]&0x0
4)>>2)+((decoded_array[i*3+1]&0x02)>>1)+(decoded_array[i*3+1]&0x01)+((decoded_array[i*3+2]&
0x20)>>5)+((decoded_array[i*3+2]&0x10)>>4)+((decoded_array[i*3+2]&0x04)>>2)+((coded_word&0
x02)>>1))%2;
   j <<= 3;
   parity_calculated = (parity_calculated|j);

j=(((decoded_array[i*3]&0x40)>>6)+((decoded_array[i*3]&0x10)>>4)+((decoded_array[i*3]&0x08)>>
3)+((decoded_array[i*3]&0x04)>>2)+(decoded_array[i*3]&0x01)+((decoded_array[i*3+1]&0x80)>>7)
+((decoded_array[i*3+1]&0x08)>>3)+((decoded_array[i*3+1]&0x04)>>2)+((decoded_array[i*3+1]&0
x02)>>1)+(decoded_array[i*3+1]&0x01)+((decoded_array[i*3+2]&0x80)>>7)+((decoded_array[i*3+2]
&0x10)>>4)+((decoded_array[i*3+2]&0x08)>>3)+((decoded_array[i*3+2]&0x02)>>1)+(coded_word&
0x01))%2;
   j <<= 2;
   parity_calculated = (parity_calculated|j);

j=(((decoded_array[i*3]&0x80)>>7)+((decoded_array[i*3]&0x20)>>5)+((decoded_array[i*3]&0x08)>>
3)+((decoded_array[i*3]&0x04)>>2)+((decoded_array[i*3]&0x02)>>1)+((decoded_array[i*3+1]&0x80
)>>7)+((decoded_array[i*3+1]&0x40)>>6)+((decoded_array[i*3+1]&0x04)>>2)+((decoded_array[i*3+
1]&0x02)>>1)+(decoded_array[i*3+1]&0x01)+((decoded_array[i*3+2]&0x80)>>7)+((decoded_array[i
*3+2]&0x40)>>6)+((decoded_array[i*3+2]&0x08)>>3)+((decoded_array[i*3+2]&0x04)>>2)+(decoded
_array[i*3+2]&0x01)+(coded_word&0x01))%2;
   j <<= 1;
   parity_calculated = (parity_calculated|j);
```

```
j=(((decoded_array[i*3]&0x20)>>5)+((decoded_array[i*3]&0x08)>>3)+((decoded_array[i*3]&0x04)>>
2)+(decoded_array[i*3]&0x01)+((decoded_array[i*3+1]&0x80)>>7)+((decoded_array[i*3+1]&0x40)>
>6)+((decoded_array[i*3+1]&0x20)>>5)+((decoded_array[i*3+1]&0x08)>>3)+((decoded_array[i*3+1]
&0x02)>>1)+((decoded_array[i*3+2]&0x20)>>5)+((decoded_array[i*3+2]&0x04)>>2)+((decoded_arra
y[i*3+2]&0x02)>>1)+(decoded_array[i*3+2]&0x01)+((coded_word&0x02)>>1))%2;
   parity_calculated = (parity_calculated|j);

   if(parity_calculated != parity_received){ /* porovnani vypoctene a prijate parity, pokud se lisi,
dochazi k oprave dekodovaneho slova */
      x = 0;
      x = (parity_calculated^parity_received);
      if(x == 1){ /* oprava bitu D30 v kodovanem poli (dulezite pro dekodovani dalsiho slova) */
         mask64 = 0x8000000000000000;
         position_array = ((i+1)*30+1)/64;
         position_bit = ((i+1)*30+1)%64;
         mask64 >>= position_bit;
         y64 = ((~coded_array[position_array])&mask64);
         coded_array[position_array] = (coded_array[position_array]&(~mask64));
         coded_array[position_array] = (coded_array[position_array]|y64);
      }else{
         mask8 = 0x01;
         check = 0;
         for(j = 0; j < 6; j++){
            check = check + (x&mask8);
            x >>= 1;
         }
         if((check%2) == 0){ /* sudy pocet chyb v parite */
            error_word = 1;
         }else{ /* lichy pocet chyb v parite */
            x = (parity_calculated^parity_received);
            x >>= 1;
            switch(x){ /* oprava dekodovaneho slova dle tabulky viz bakalarska prace */
            case 1: /* bit D29 */
               mask64 = 0x8000000000000000;
               position_array = ((i+1)*30)/64;
               position_bit = ((i+1)*30)%64;
               mask64 >>= position_bit;
               y64 = ((~coded_array[position_array])&mask64);
               coded_array[position_array] = (coded_array[position_array]&(~mask64));
               coded_array[position_array] = (coded_array[position_array]|y64);
               break;
            case 2: /* bit D28 */
               break;
            case 3: /* bit d9 */
               mask8 = 0x80;
               y = ((~decoded_array[i*3+1])&mask8);
               decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
```

```
    decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
    break;
case 4: /* bit D27 */
    break;
case 5: /* bit d22 */
    mask8 = 0x04;
    y = ((~decoded_array[i*3+2])&mask8);
    decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
    decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
    break;
case 6: /* bit d8 */
    mask8 = 0x01;
    y = ((~decoded_array[i*3])&mask8);
    decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
case 7: /* bit d16 */
    mask8 = 0x01;
    y = ((~decoded_array[i*3+1])&mask8);
    decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
    decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
    break;
case 8: /* bit D26 */
    break;
case 9: /* bit d24 */
    mask8 = 0x01;
    y = ((~decoded_array[i*3+2])&mask8);
    decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
    decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
    break;
case 11: /* bit d21 */
    mask8 = 0x08;
    y = ((~decoded_array[i*3+2])&mask8);
    decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
    decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
    break;
case 12: /* bit d19 */
    mask8 = 0x20;
    y = ((~decoded_array[i*3+2])&mask8);
    decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
    decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
    break;
case 13: /* bit d7 */
    mask8 = 0x02;
    y = ((~decoded_array[i*3])&mask8);
    decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
```

```
case 14: /* bit d4 */
   mask8 = 0x10;
   y = ((~decoded_array[i*3])&mask8);
   decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
   decoded_array[i*3] = (decoded_array[i*3]|y);
   break;
case 15: /* bit d15 */
   mask8 = 0x02;
   y = ((~decoded_array[i*3+1])&mask8);
   decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
   decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
   break;
case 16: /* bit D25 */
   break;
case 17: /* bit d10 */
   mask8 = 0x40;
   y = ((~decoded_array[i*3+1])&mask8);
   decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
   decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
   break;
case 18: /* bit d23 */
   mask8 = 0x02;
   y = ((~decoded_array[i*3+2])&mask8);
   decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
   decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
   break;
case 19: /* bit d17 */
   mask8 = 0x80;
   y = ((~decoded_array[i*3+2])&mask8);
   decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
   decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
   break;
case 21: /* bit d1 */
   mask8 = 0x80;
   y = ((~decoded_array[i*3])&mask8);
   decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
   decoded_array[i*3] = (decoded_array[i*3]|y);
   break;
case 22: /* bit d20 */
   mask8 = 0x10;
   y = ((~decoded_array[i*3+2])&mask8);
   decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
   decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
   break;
case 23: /* bit d5 */
   mask8 = 0x08;
   y = ((~decoded_array[i*3])&mask8);
   decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
```

```
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
case 24: /* bit d11 */
    mask8 = 0x20;
    y = ((~decoded_array[i*3+1])&mask8);
    decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
    decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
    break;
case 25: /* bit d18 */
    mask8 = 0x40;
    y = ((~decoded_array[i*3+2])&mask8);
    decoded_array[i*3+2] = (decoded_array[i*3+2]&(~mask8));
    decoded_array[i*3+2] = (decoded_array[i*3+2]|y);
    break;
case 26: /* bit d2 */
    mask8 = 0x40;
    y = ((~decoded_array[i*3])&mask8);
    decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
case 27: /* bit d6 */
    mask8 = 0x04;
    y = ((~decoded_array[i*3])&mask8);
    decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
case 28: /* bit d12 */
    mask8 = 0x10;
    y = ((~decoded_array[i*3+1])&mask8);
    decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
    decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
    break;
case 29: /* bit d3 */
    mask8 = 0x20;
    y = ((~decoded_array[i*3])&mask8);
    decoded_array[i*3] = (decoded_array[i*3]&(~mask8));
    decoded_array[i*3] = (decoded_array[i*3]|y);
    break;
case 30: /* bit d13 */
    mask8 = 0x08;
    y = ((~decoded_array[i*3+1])&mask8);
    decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
    decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
    break;
case 31: /* bit d14 */
    mask8 = 0x04;
    y = ((~decoded_array[i*3+1])&mask8);
    decoded_array[i*3+1] = (decoded_array[i*3+1]&(~mask8));
```

```
            decoded_array[i*3+1] = (decoded_array[i*3+1]|y);
            break;
          default: /* v dekodovanem slove je vic nez 1 chyba */
            error_word = 1;
          }
        }
      }
    }
}

return error_word; /* error_word = 1 -> v podramci se nachazi minimalne jedno chybove slovo */
}

/* vypocet promennych v jednotlivych podramcich (strukturach) */
void data_subframe1(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
sub1[SV_number-1].WN = calculate(48,10,9);
sub1[SV_number-1].kod_L2 = calculate(58,2,1);
sub1[SV_number-1].URA_index = calculate(60,4,3);
sub1[SV_number-1].SV_health = calculate(64,6,5);
sub1[SV_number-1].IODC = (calculate(70,2,9) + calculate(168,8,7));
sub1[SV_number-1].L2 = calculate(72,1,0);
sub1[SV_number-1].TGD = calculate_binary(160,8,-24);
sub1[SV_number-1].toc = calculate(176,16,19);
sub1[SV_number-1].af2 = calculate_binary(192,8,-48);
sub1[SV_number-1].af1 = calculate_binary(200,16,-28);
sub1[SV_number-1].af0 = calculate_binary(216,22,-10);
}

void data_subframe2(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
sub2[SV_number-1].IODE = calculate(48,8,7);
sub2[SV_number-1].Crs = calculate_binary(56,16,10);
sub2[SV_number-1].delta_n = calculate_binary(72,16,-28);
sub2[SV_number-1].M0 = calculate_binary(88,32,0);
sub2[SV_number-1].Cuc = calculate_binary(120,16,-14);
sub2[SV_number-1].e = calculate(136,32,-2);
sub2[SV_number-1].Cus = calculate_binary(168,16,-14);
sub2[SV_number-1].odm_A = calculate(184,32,12);
sub2[SV_number-1].toe = calculate(216,16,19);
sub2[SV_number-1].fit_interval_flag = calculate(232,1,0);
sub2[SV_number-1].AODO = calculate(233,5,4);
}

void data_subframe3(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
sub3[SV_number-1].Cic = calculate_binary(48,16,-14);
sub3[SV_number-1].OMG0 = calculate_binary(64,32,0);
```

```
sub3[SV_number-1].Cis = calculate_binary(96,16,-14);
sub3[SV_number-1].i0 = calculate_binary(112,32,0);
sub3[SV_number-1].Crc = calculate_binary(144,16,10);
sub3[SV_number-1].omg = calculate_binary(160,32,0);
sub3[SV_number-1].OMGd = calculate_binary(192,24,-20);
sub3[SV_number-1].IODE = calculate(216,8,7);
sub3[SV_number-1].idot = calculate_binary(224,14,-30);
}

void data_subframe4and5(){
sub4and5.data_ID = calculate(48,2,1);
sub4and5.SV_ID = calculate(50,6,5);
sub4and5.e = calculate(56,16,-6);
sub4and5.toa = calculate(72,8,19);
sub4and5.delta_i = calculate_binary(80,16,-4);
sub4and5.OMGd = calculate_binary(96,16,-23);
sub4and5.SV_health = calculate(112,8,7);
sub4and5.odm_A = calculate(120,24,12);
sub4and5.OMG0 = calculate_binary(144,24,0);
sub4and5.omg = calculate_binary(168,24,0);
sub4and5.M0 = calculate_binary(192,24,0);
sub4and5.af0 = (calculate_binary(216,8,-10) + calculate(235,3,-18));
sub4and5.af1 = calculate_binary(224,11,-28);
}

void data_subframe4page18(){
sub4p18.data_ID = calculate(48,2,1);
sub4p18.SV_ID = calculate(50,6,5);
sub4p18.alfa0 = calculate_binary(56,8,-23);
sub4p18.alfa1 = calculate_binary(64,8,-20);
sub4p18.alfa2 = calculate_binary(72,8,-17);
sub4p18.alfa3 = calculate_binary(80,8,-17);
sub4p18.beta0 = calculate_binary(88,8,18);
sub4p18.beta1 = calculate_binary(96,8,21);
sub4p18.beta2 = calculate_binary(104,8,23);
sub4p18.beta3 = calculate_binary(112,8,23);
sub4p18.A1 = calculate_binary(120,24,-27);
sub4p18.A0 = calculate_binary(144,32,1);
sub4p18.tot = calculate(176,8,19);
sub4p18.WNt = calculate(184,8,7);
sub4p18.delta_tLS = calculate_binary(192,8,7);
sub4p18.WNLSF = calculate(200,8,7);
sub4p18.DN = calculate(208,8,7);
sub4p18.delta_tLSF = calculate_binary(216,8,7);
}

void data_subframe4page25(){
uint8_t i;
```

```
sub4p25.data_ID = calculate(48,2,1);
sub4p25.SV_ID = calculate(50,6,5);
for(i = 0; i < 32; i++){
   sub4p25.AS_config[i] = calculate((56+i*4),4,3);
}
for(i = 0; i < 8; i++){
   sub4p25.SV_health[i] = calculate((186+i*6),6,5);
}
}

void data_subframe5page25(){
uint8_t i;
sub5p25.data_ID = calculate(48,2,1);
sub5p25.SV_ID = calculate(50,6,5);
sub5p25.toa = calculate(56,8,19);
sub5p25.WNa = calculate(64,8,7);
for(i = 0; i < 24; i++){
   sub5p25.SV_health[i] = calculate((72+i*6),6,5);
}
}

/* aktualizace promennych v jednotlivych podramcich (strukturach) */
void compare_subframe1(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
uint8_t x = 0;
if(sub1[SV_number-1].WN == sub1_act[SV_number-1].WN)
   x++;
if(sub1[SV_number-1].kod_L2 == sub1_act[SV_number-1].kod_L2)
   x++;
if(sub1[SV_number-1].URA_index == sub1_act[SV_number-1].URA_index)
   x++;
if(sub1[SV_number-1].SV_health == sub1_act[SV_number-1].SV_health)
   x++;
if(sub1[SV_number-1].IODC == sub1_act[SV_number-1].IODC)
   x++;
if(sub1[SV_number-1].L2 == sub1_act[SV_number-1].L2)
   x++;
if(sub1[SV_number-1].TGD == sub1_act[SV_number-1].TGD)
   x++;
if(sub1[SV_number-1].toc == sub1_act[SV_number-1].toc)
   x++;
if(sub1[SV_number-1].af0 == sub1_act[SV_number-1].af0)
   x++;
if(sub1[SV_number-1].af1 == sub1_act[SV_number-1].af1)
   x++;
if(sub1[SV_number-1].af2 == sub1_act[SV_number-1].af2)
   x++;
if(x < 11){
```

```
   sub1_2act[SV_number-1].WN = sub1_act[SV_number-1].WN;
   sub1_2act[SV_number-1].kod_L2 = sub1_act[SV_number-1].kod_L2;
   sub1_2act[SV_number-1].URA_index = sub1_act[SV_number-1].URA_index;
   sub1_2act[SV_number-1].SV_health = sub1_act[SV_number-1].SV_health;
   sub1_2act[SV_number-1].IODC = sub1_act[SV_number-1].IODC;
   sub1_2act[SV_number-1].L2 = sub1_act[SV_number-1].L2;
   sub1_2act[SV_number-1].TGD = sub1_act[SV_number-1].TGD;
   sub1_2act[SV_number-1].toc = sub1_act[SV_number-1].toc;
   sub1_2act[SV_number-1].af0 = sub1_act[SV_number-1].af0;
   sub1_2act[SV_number-1].af1 = sub1_act[SV_number-1].af1;
   sub1_2act[SV_number-1].af2 = sub1_act[SV_number-1].af2;
   sub1_act[SV_number-1].WN = sub1[SV_number-1].WN;
   sub1_act[SV_number-1].kod_L2 = sub1[SV_number-1].kod_L2;
   sub1_act[SV_number-1].URA_index = sub1[SV_number-1].URA_index;
   sub1_act[SV_number-1].SV_health = sub1[SV_number-1].SV_health;
   sub1_act[SV_number-1].IODC = sub1[SV_number-1].IODC;
   sub1_act[SV_number-1].L2 = sub1[SV_number-1].L2;
   sub1_act[SV_number-1].TGD = sub1[SV_number-1].TGD;
   sub1_act[SV_number-1].toc = sub1[SV_number-1].toc;
   sub1_act[SV_number-1].af0 = sub1[SV_number-1].af0;
   sub1_act[SV_number-1].af1 = sub1[SV_number-1].af1;
   sub1_act[SV_number-1].af2 = sub1[SV_number-1].af2;

   new_data_subframe = 1;
}
}

void compare_subframe2(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
uint8_t x = 0;
if(sub2[SV_number-1].IODE == sub2_act[SV_number-1].IODE)
   x++;
if(sub2[SV_number-1].Crs == sub2_act[SV_number-1].Crs)
   x++;
if(sub2[SV_number-1].delta_n == sub2_act[SV_number-1].delta_n)
   x++;
if(sub2[SV_number-1].M0 == sub2_act[SV_number-1].M0)
   x++;
if(sub2[SV_number-1].Cuc == sub2_act[SV_number-1].Cuc)
   x++;
if(sub2[SV_number-1].e == sub2_act[SV_number-1].e)
   x++;
if(sub2[SV_number-1].Cus == sub2_act[SV_number-1].Cus)
   x++;
if(sub2[SV_number-1].odm_A == sub2_act[SV_number-1].odm_A)
   x++;
if(sub2[SV_number-1].toe == sub2_act[SV_number-1].toe)
   x++;
```

```
if(sub2[SV_number-1].fit_interval_flag == sub2_act[SV_number-1].fit_interval_flag)
   x++;
if(sub2[SV_number-1].AODO == sub2_act[SV_number-1].AODO)
   x++;
if(x < 11){
   sub2_2act[SV_number-1].IODE = sub2_act[SV_number-1].IODE;
   sub2_2act[SV_number-1].Crs = sub2_act[SV_number-1].Crs;
   sub2_2act[SV_number-1].delta_n = sub2_act[SV_number-1].delta_n;
   sub2_2act[SV_number-1].M0 = sub2_act[SV_number-1].M0;
   sub2_2act[SV_number-1].Cuc = sub2_act[SV_number-1].Cuc;
   sub2_2act[SV_number-1].e = sub2_act[SV_number-1].e;
   sub2_2act[SV_number-1].Cus = sub2_act[SV_number-1].Cus;
   sub2_2act[SV_number-1].odm_A = sub2_act[SV_number-1].odm_A;
   sub2_2act[SV_number-1].toe = sub2_act[SV_number-1].toe;
   sub2_2act[SV_number-1].fit_interval_flag = sub2_act[SV_number-1].fit_interval_flag;
   sub2_2act[SV_number-1].AODO = sub2_act[SV_number-1].AODO;
   sub2_act[SV_number-1].IODE = sub2[SV_number-1].IODE;
   sub2_act[SV_number-1].Crs = sub2[SV_number-1].Crs;
   sub2_act[SV_number-1].delta_n = sub2[SV_number-1].delta_n;
   sub2_act[SV_number-1].M0 = sub2[SV_number-1].M0;
   sub2_act[SV_number-1].Cuc = sub2[SV_number-1].Cuc;
   sub2_act[SV_number-1].e = sub2[SV_number-1].e;
   sub2_act[SV_number-1].Cus = sub2[SV_number-1].Cus;
   sub2_act[SV_number-1].odm_A = sub2[SV_number-1].odm_A;
   sub2_act[SV_number-1].toe = sub2[SV_number-1].toe;
   sub2_act[SV_number-1].fit_interval_flag = sub2[SV_number-1].fit_interval_flag;
   sub2_act[SV_number-1].AODO = sub2[SV_number-1].AODO;

   new_data_subframe = 2;
}
}

void compare_subframe3(uint8_t SV_number){ /* par: promenna SV_number predana do fce
process_GPS_data */
uint8_t x = 0;
if(sub3[SV_number-1].Cic == sub3_act[SV_number-1].Cic)
   x++;
if(sub3[SV_number-1].OMG0 == sub3_act[SV_number-1].OMG0)
   x++;
if(sub3[SV_number-1].Cis == sub3_act[SV_number-1].Cis)
   x++;
if(sub3[SV_number-1].i0 == sub3_act[SV_number-1].i0)
   x++;
if(sub3[SV_number-1].Crc == sub3_act[SV_number-1].Crc)
   x++;
if(sub3[SV_number-1].omg == sub3_act[SV_number-1].omg)
   x++;
if(sub3[SV_number-1].OMGd == sub3_act[SV_number-1].OMGd)
```

```
   x++;
if(sub3[SV_number-1].IODE == sub3_act[SV_number-1].IODE)
   x++;
if(sub3[SV_number-1].idot == sub3_act[SV_number-1].idot)
   x++;
if(x < 9){
   sub3_2act[SV_number-1].Cic = sub3_act[SV_number-1].Cic;
   sub3_2act[SV_number-1].OMG0 = sub3_act[SV_number-1].OMG0;
   sub3_2act[SV_number-1].Cis = sub3_act[SV_number-1].Cis;
   sub3_2act[SV_number-1].i0 = sub3_act[SV_number-1].i0;
   sub3_2act[SV_number-1].Crc = sub3_act[SV_number-1].Crc;
   sub3_2act[SV_number-1].omg = sub3_act[SV_number-1].omg;
   sub3_2act[SV_number-1].OMGd = sub3_act[SV_number-1].OMGd;
   sub3_2act[SV_number-1].IODE = sub3_act[SV_number-1].IODE;
   sub3_2act[SV_number-1].idot = sub3_act[SV_number-1].idot;
   sub3_act[SV_number-1].Cic = sub3[SV_number-1].Cic;
   sub3_act[SV_number-1].OMG0 = sub3[SV_number-1].OMG0;
   sub3_act[SV_number-1].Cis = sub3[SV_number-1].Cis;
   sub3_act[SV_number-1].i0 = sub3[SV_number-1].i0;
   sub3_act[SV_number-1].Crc = sub3[SV_number-1].Crc;
   sub3_act[SV_number-1].omg = sub3[SV_number-1].omg;
   sub3_act[SV_number-1].OMGd = sub3[SV_number-1].OMGd;
   sub3_act[SV_number-1].IODE = sub3[SV_number-1].IODE;
   sub3_act[SV_number-1].idot = sub3[SV_number-1].idot;

   new_data_subframe = 3;
}
}

void compare_subframe4and5(){
uint8_t x = 0;
if(sub4and5.data_ID == sub4and5_act.data_ID)
   x++;
if(sub4and5.SV_ID == sub4and5_act.SV_ID)
   x++;
if(sub4and5.e == sub4and5_act.e)
   x++;
if(sub4and5.toa == sub4and5_act.toa)
   x++;
if(sub4and5.delta_i == sub4and5_act.delta_i)
   x++;
if(sub4and5.OMGd == sub4and5_act.OMGd)
   x++;
if(sub4and5.SV_health == sub4and5_act.SV_health)
   x++;
if(sub4and5.odm_A == sub4and5_act.odm_A)
   x++;
if(sub4and5.OMG0 == sub4and5_act.OMG0)
```

```
   x++;
if(sub4and5.omg == sub4and5_act.omg)
   x++;
if(sub4and5.M0 == sub4and5_act.M0)
   x++;
if(sub4and5.af0 == sub4and5_act.af0)
   x++;
if(sub4and5.af1 == sub4and5_act.af1)
   x++;
if(x < 13){
   sub4and5_2act.data_ID = sub4and5_act.data_ID;
   sub4and5_2act.SV_ID = sub4and5_act.SV_ID;
   sub4and5_2act.e = sub4and5_act.e;
   sub4and5_2act.toa = sub4and5_act.toa;
   sub4and5_2act.delta_i = sub4and5_act.delta_i;
   sub4and5_2act.OMGd = sub4and5_act.OMGd;
   sub4and5_2act.SV_health = sub4and5_act.SV_health;
   sub4and5_2act.odm_A = sub4and5_act.odm_A;
   sub4and5_2act.OMG0 = sub4and5_act.OMG0;
   sub4and5_2act.omg = sub4and5_act.omg;
   sub4and5_2act.M0 = sub4and5_act.M0;
   sub4and5_2act.af0 = sub4and5_act.af0;
   sub4and5_2act.af1 = sub4and5_act.af1;
   sub4and5_act.data_ID = sub4and5.data_ID;
   sub4and5_act.SV_ID = sub4and5.SV_ID;
   sub4and5_act.e = sub4and5.e;
   sub4and5_act.toa = sub4and5.toa;
   sub4and5_act.delta_i = sub4and5.delta_i;
   sub4and5_act.OMGd = sub4and5.OMGd;
   sub4and5_act.SV_health = sub4and5.SV_health;
   sub4and5_act.odm_A = sub4and5.odm_A;
   sub4and5_act.OMG0 = sub4and5.OMG0;
   sub4and5_act.omg = sub4and5.omg;
   sub4and5_act.M0 = sub4and5.M0;
   sub4and5_act.af0 = sub4and5.af0;
   sub4and5_act.af1 = sub4and5.af1;

   new_data_subframe = 4;
}
}

void compare_subframe4page18(){
uint8_t x = 0;
if(sub4p18.data_ID == sub4p18_act.data_ID)
   x++;
if(sub4p18.SV_ID == sub4p18_act.SV_ID)
   x++;
if(sub4p18.alfa0 == sub4p18_act.alfa0)
```

```
   x++;
if(sub4p18.alfa1 == sub4p18_act.alfa1)
   x++;
if(sub4p18.alfa2 == sub4p18_act.alfa2)
   x++;
if(sub4p18.alfa3 == sub4p18_act.alfa3)
   x++;
if(sub4p18.beta0 == sub4p18_act.beta0)
   x++;
if(sub4p18.beta1 == sub4p18_act.beta1)
   x++;
if(sub4p18.beta2 == sub4p18_act.beta2)
   x++;
if(sub4p18.beta3 == sub4p18_act.beta3)
   x++;
if(sub4p18.A1 == sub4p18_act.A1)
   x++;
if(sub4p18.A0 == sub4p18_act.A0)
   x++;
if(sub4p18.tot == sub4p18_act.tot)
   x++;
if(sub4p18.WNt == sub4p18_act.WNt)
   x++;
if(sub4p18.delta_tLS == sub4p18_act.delta_tLS)
   x++;
if(sub4p18.WNLSF == sub4p18_act.WNLSF)
   x++;
if(sub4p18.DN == sub4p18_act.DN)
   x++;
if(sub4p18.delta_tLSF == sub4p18_act.delta_tLSF)
   x++;
if(x < 18){
   sub4p18_2act.data_ID = sub4p18_act.data_ID;
   sub4p18_2act.SV_ID = sub4p18_act.SV_ID;
   sub4p18_2act.alfa0 = sub4p18_act.alfa0;
   sub4p18_2act.alfa1 = sub4p18_act.alfa1;
   sub4p18_2act.alfa2 = sub4p18_act.alfa2;
   sub4p18_2act.alfa3 = sub4p18_act.alfa3;
   sub4p18_2act.beta0 = sub4p18_act.beta0;
   sub4p18_2act.beta1 = sub4p18_act.beta1;
   sub4p18_2act.beta2 = sub4p18_act.beta2;
   sub4p18_2act.beta3 = sub4p18_act.beta3;
   sub4p18_2act.A1 = sub4p18_act.A1;
   sub4p18_2act.A0 = sub4p18_act.A0;
   sub4p18_2act.tot = sub4p18_act.tot;
   sub4p18_2act.WNt = sub4p18_act.WNt;
   sub4p18_2act.delta_tLS = sub4p18_act.delta_tLS;
   sub4p18_2act.WNLSF = sub4p18_act.WNLSF;
```

```
   sub4p18_2act.DN = sub4p18_act.DN;
   sub4p18_2act.delta_tLSF = sub4p18_act.delta_tLSF;
   sub4p18_act.data_ID = sub4p18.data_ID;
   sub4p18_act.SV_ID = sub4p18.SV_ID;
   sub4p18_act.alfa0 = sub4p18.alfa0;
   sub4p18_act.alfa1 = sub4p18.alfa1;
   sub4p18_act.alfa2 = sub4p18.alfa2;
   sub4p18_act.alfa3 = sub4p18.alfa3;
   sub4p18_act.beta0 = sub4p18.beta0;
   sub4p18_act.beta1 = sub4p18.beta1;
   sub4p18_act.beta2 = sub4p18.beta2;
   sub4p18_act.beta3 = sub4p18.beta3;
   sub4p18_act.A1 = sub4p18.A1;
   sub4p18_act.A0 = sub4p18.A0;
   sub4p18_act.tot = sub4p18.tot;
   sub4p18_act.WNt = sub4p18.WNt;
   sub4p18_act.delta_tLS = sub4p18.delta_tLS;
   sub4p18_act.WNLSF = sub4p18.WNLSF;
   sub4p18_act.DN = sub4p18.DN;
   sub4p18_act.delta_tLSF = sub4p18.delta_tLSF;

   new_data_subframe = 5;
}
}

void compare_subframe4page25(){
uint8_t x = 0;
uint8_t i;
if(sub4p25.data_ID == sub4p25_act.data_ID)
   x++;
if(sub4p25.SV_ID == sub4p25_act.SV_ID)
   x++;
for(i = 0; i < 32; i++){
   if(sub4p25.AS_config[i] == sub4p25_act.AS_config[i])
      x++;
}
for(i = 0; i < 8; i++){
   if(sub4p25.SV_health[i] == sub4p25_act.SV_health[i])
      x++;
}
if(x < 42){
   sub4p25_2act.data_ID = sub4p25_act.data_ID;
   sub4p25_2act.SV_ID = sub4p25_act.SV_ID;
   for(i = 0; i < 32; i++)
      sub4p25_2act.AS_config[i] = sub4p25_act.AS_config[i];
   for(i = 0; i > 8; i++)
      sub4p25_2act.SV_health[i] = sub4p25_act.SV_health[i];
   sub4p25_act.data_ID = sub4p25.data_ID;
```

```
   sub4p25_act.SV_ID = sub4p25.SV_ID;
   for(i = 0; i < 32; i++)
      sub4p25_act.AS_config[i] = sub4p25.AS_config[i];
   for(i = 0; i < 8; i++)
      sub4p25_act.SV_health[i] = sub4p25.SV_health[i];

   new_data_subframe = 6;
}
}

void compare_subframe5page25(){
uint8_t x = 0;
uint8_t i;
if(sub5p25.data_ID == sub5p25_act.data_ID)
   x++;
if(sub5p25.SV_ID == sub5p25_act.SV_ID)
   x++;
if(sub5p25.toa == sub5p25_act.toa)
   x++;
if(sub5p25.WNa == sub5p25_act.WNa)
   x++;
for(i = 0; i < 24; i++){
   if(sub5p25.SV_health[i] == sub5p25_act.SV_health[i])
      x++;
}
if(x < 28){
   sub5p25_2act.data_ID = sub5p25_act.data_ID;
   sub5p25_2act.SV_ID = sub5p25_act.SV_ID;
   sub5p25_2act.toa = sub5p25_act.toa;
   sub5p25_2act.WNa = sub5p25_act.WNa;
   for(i = 0; i < 24; i++)
      sub5p25_2act.SV_health[i] = sub5p25_act.SV_health[i];
   sub5p25_act.data_ID = sub5p25.data_ID;
   sub5p25_act.SV_ID = sub5p25.SV_ID;
   sub5p25_act.toa = sub5p25.toa;
   sub5p25_act.WNa = sub5p25.WNa;
   for(i = 0; i < 24; i++)
      sub5p25_act.SV_health[i] = sub5p25.SV_health[i];

   new_data_subframe = 7;
}
}

/* trideni podramcu (dat) */
void separate(uint8_t SV_number){ /* par: promenna SV_number predana do fce process_GPS_data
*/
uint8_t switch_case;
uint8_t subframe;
```

```
uint8_t SV_ID;

subframe = calculate(43,3,2); /* vypocet cisla podramce */
SV_ID = calculate(50,6,5); /* vypocet SV_ID */

if(subframe <= 3){ /* podramce 1, 2 a 3 */
   switch_case = subframe;
}
else if(subframe == 4){ /* podramec 4 */
   if(SV_ID == 63){ /* 25. stranka */
      switch_case = 6;
   }
   else if(SV_ID == 56){ /* 18. stranka */
      switch_case = 5;
   }
   else if(SV_ID >= 52 && SV_ID <= 62 && SV_ID != 56){
      switch_case = 8;
   }
   else{ /* stranky 2 - 5 a 7 - 10 */
      switch_case = 4;
   }
}
else if(subframe == 5){ /* podramec 5 */
   if(SV_ID == 51){
      switch_case = 7; /* 25. stranka */
   }else{
      switch_case = 4; /* 1. - 24. stranka */
   }
}
else{ /* jina, chybna varianta */
   switch_case = 9;
}

switch(switch_case){
case 0:
   #ifdef TEST_data
   xil_printf("Neplatny vstup! \n");
   #endif
   break;
case 1:
   data_subframe1(SV_number);
   compare_subframe1(SV_number);
   break;
case 2:
   data_subframe2(SV_number);
   compare_subframe2(SV_number);
   break;
case 3:
```

```
   data_subframe3(SV_number);
   compare_subframe3(SV_number);
   break;
case 4:
   data_subframe4and5();
   compare_subframe4and5();
   break;
case 5:
   data_subframe4page18();
   compare_subframe4page18();
   break;
case 6:
   data_subframe4page25();
   compare_subframe4page25();
   break;
case 7:
   data_subframe5page25();
   compare_subframe5page25();
   break;
case 8:
   #ifdef TEST_data
   xil_printf("Tento podramec obsahuje pouze rezervovane nebo nevyuzite bity, tj. neobsahuje data.
\n");
   #endif
   break;
default:
   #ifdef TEST_data
   xil_printf("Neplatny vstup! \n");
   #endif
}
}

/* zpracovani dat z nav. zpravy GPS */
int32_t process_GPS_data(uint64_t * buffer, uint8_t SV_number){ /* par: odkaz na buffer s
demodulovanymi bity, cislo druzice */
uint8_t is_preamble;
int32_t zcount;
is_preamble = find_preamble(buffer); /* vyhledavani preambule */
if(is_preamble == 1){
   uint64_t memory[5]; /* definice mezipameti */
   uint64_t bits;
   uint8_t i, error_word;
   uint8_t mask = 0x80;
   uint8_t zcount_array[3];
   for(i = 0; i < 5; i++){ /* naplneni mezipameti kodovanym podramcem */
      memory[i] = buffer[4-i];
      memory[i] <<= 10;
      if(i != 4){
```

```
            bits = buffer[4-i-1];
            bits >>= 54;
            memory[i] = (memory[i]|bits);
        }
    }
    error_word = decode(memory); /* dekodovani podramce */

    zcount = calculate(24,17,16); /* vypocet z-countu */

    if(error_word == 0){ /* trideni podramce, pokud neobsahuje chybove slovo */
        separate(SV_number);
        }
}else{
    zcount = -1;
}
return zcount; /* z-count = -1 -> nezasynchronizovan podramec, z-count >= 0 -> pravdepodobne
zasynchronizovan podramec */
}
```