VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ŘÍDICÍ JEDNOTKA MONTÁŽE ASTRONOMICKÉHO DALEKOHLEDU

CONTROL UNIT FOR TELESCOPE MOUNT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN CAGAŠ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Cagaš Martin**
Program: Informační technologie
Název: **Řídicí jednotka montáže astronomického dalekohledu**
Control Unit for Telescope Mount
Kategorie: Vestavěné systémy
Zadání:

1. Seznamte se s běžně používanými montážemi astronomického dalekohledu pro podporu astrofotografie umožňujícími v reálném čase sledovat trajektorii objektů na obloze. Pozornost věnujte zejména způsobu sledování objektů na obloze a souvisejícímu řízení.
2. Pro zvolenou řídicí jednotku navrhněte firmware umožňující řídit krokové motory na základě znalosti polohy montáže získané z enkodérů a parametrů zadaných uživatelem. Cílem je vytvořit autonomní jednotku, kterou je možné řídit povely zasílanými z PC.
3. Zpracujte studii na výše uvedené téma.
4. Navržený firmware implementujte, ověřte jeho funkčnost a vyhodnoťte dosažené parametry. Snažte se o maximální efektivnost a flexibilitu navrženého řešení.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

Abstrakt

Tato práce rozebírá základní mechanismy a jejich implementaci při realizaci firmware řídicí jednotky montáže astronomického dalekohledu. Cílem je vytvořit samostatnou jednotku podporující potřebné funkce pro využití ve vědecké astronomii a astrofotografii. Jednotka je postavena na kombinaci mikrokontroléru realizujícího náročné výpočetní úkony, a FPGA mající na starost časově kritické funkce. Jednotka využívá vlastního algoritmu pro určování aktuálních souřadnic na nebeské sféře. Součástí práce je definice vlastního protokolu pro komunikaci s připojeným PC. Výsledkem práce je vestavěný systém zprostředkovávající základní funkce potřebné pro použití v astronomii s prostorem pro rozšíření o dodatečnou funkcionalitu. Přínosem práce je zapojení FPGA do problematiky jinak běžně řešené mikrokontroléry.

Abstract

This thesis examines the basic mechanisms and their implementation in creation of a firmware for control units of telescope mounts. The goal of this thesis is to create an autonomous unit supporting all the necessary functions required for scientific astronomy and astrophotography. The unit is based on the combination of a microcontroller, performing complex computational tasks, and FPGA performing time-critical operations. The unit utilizes its own algorithm for determining the current astronomical coordinates on the celestial sphere. A part of the thesis is a definition of own protocol for communication with a connected PC. The outcome of the thesis is an embedded system offering the basic required functionality for astronomical imaging with the space for implementing additional features. The benefit of the thesis is utilization of an FPGA in a topic usually solved by microcontrollers.

Klíčová slova

vestavěné systémy, mcu, c, fpga, vhdl, ovládání motorů, řídící jednotka, astronomie, astrofotografie

Keywords

embedded systems, mcu, c, fpga, vhdl, driving motors, control unit, astronomy, astrophotography

Citace

CAGAŠ, Martin. *Řídicí jednotka montáže astronomického dalekohledu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Zdeněk Vašíček, Ph.D.

Řídicí jednotka montáže astronomického dalekohledu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana docenta inženýra Zdeňka Vašíčka, Ph.D.. Další technické informace mi poskytli zaměstnanci firmy Moravské Přístroje a. s. a pan inženýr Pavel Cagaš, Ph.D., který mi poskytl své zkušenosti s astronomií a astronomickým pozorováním. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Cagaš
14. května 2019

Poděkování

V prvé řadě bych rád poděkoval panu docentu inženýrovi Zdeňku Vašíčkovi, Ph.D. za to, že mi umožnil práci dokončit i v následujícím roce a za všechny technické poznatky a přínosy, které byly při řešení nedocenitelné. Také bych rád poděkoval panu inženýru Pavlu Cagašovi, Ph.D. za konzultaci astronomických témat a za všechny poskytnuté vlastní zkušenosti s astronomií a astronomickým pozorováním.

Obsah

Seznam tabulek

Kapitola 1

Úvod

S pokrokem lidských vědomostí a technologie je získávání nových informací čím dál tím složitější. Když nové informace přestávají být získatelné běžnými prostředky, jako jednoduchým pozorováním a ručním měřením, je potřeba zapojení nejnovější technologií do procesu, aby bylo možné vůbec nové informace získat.

Astronomie není výjimkou. Doba pozorování okem a pojmenovávání svítících bodů na obloze už je dávnou minulostí. První ruční použití dalekohledů umožnilo odhalit nové objekty na hvězdné obloze, jako měsíce ostatních planet, či mlhoviny a hvězdokupy. Pro zapojení obrazové techniky však již ruční řízení nestačí. Ať už světlo zachytává fotografický film, či CCD nebo CMOS čip digitální kamery, krátká expozice nestačí a je potřeba udržet přesné zorné pole v objektivu dalekohledu.

Na řadu přichází automatizace pohybu montáže, která dalekohled vytvářející snímky oblohy nese, pomocí elektromotorů a elektroniky ovládající tyto motory. Elektronika z počátku zprostředkovávající jednoduchý lineární pohyb v rektascenzi byla později rozšířena o další funkce s přibývajícími požadavky—další sledovací rychlosti pro snímkování Slunce a Měsíce, motorizovaný pohyb v obou osách různými rychlostmi řízený ručním ovladačem, automatické najetí na zadané hvězdné souřadnice, automatická korekce odchylek chodu montáže prostřednictvím zpětné vazby zprostředkované tzv. "pointační"kamerou, překládání tubusu vzhledem k pilíři u Německých montáží (GEM—German Equatorial Mount) a mnoho dalšího. To umožnilo zefektivnit a dále prohloubit možnosti astronomického pozorování a lidského poznání vesmíru.

Cílem práce bylo vytvořit firmware pro řídicí elektroniku jednotky—ne pouze teoreticky navrhnout, ale v kombinaci s již existujícím hardware vytvořit samostatnou jednotku podporující astronomické vědecké měření a astrofotografii.

Kapitola 2 rozebírá základní astronomické koncepty na kterých je práce postavena pro snadnější porozumění zvolenému řešení. Kapitola 3 se zabývá teoretickým návrhem veškeré funkcionality potřebné pro dosáhnutí požadované funkčnosti. V kapitole 4 je rozebírána problematika návrhu kódu pro FPGA využité v jednotce, ale i důvody pro volbu FPGA oproti mikrokontroléru a výhody, které FPGA přináší. Na závěr kapitola 5 rozebere implementaci kódu mikrokontroléru, jeho zasazení do systému a na základě informací z předchozích kapitol i jeho interakci s ostatními částmi jednotky. Závěr práce 6 obsahuje nejen zhodnocení práce, ale také rozebírá budoucí rozšíření práce pro vytvoření skutečně reálně použitého produktu.

Kapitola 2

Astronomie a její využití v práci

2.1 Význam astronomie v historii a dnes

Samotné slovo astronomie pochází z řeckého slova "astronomía", znamenajícího "uspořádávání či kategorizace hvězd". Ve starověkém Řecku byly základy této vědy položeny, i když pozorování oblohy a zaznamenávání dějů na ní se věnovaly téměř všechny starověké národy od starověké Číny, přes Egypt a Řecko až po civilizace v Jižní Americe.

Pozorování dějů na obloze a vysledování závislostí v pohybech nebeských těles umožnilo lidem měřit čas a sestavit kalendář. Podle kalendáře pak bylo možné plánovat zemědělské práce, poloha nebeských objektů umožňovala navigovat při námořních plavbách, apod. Zejména ale astronomická pozorování dovolovala pochopit a poznat svět ve kterém žijeme, od postupného poznání vzdáleností a oběžných drah planet až po dnešní porozumění stavbě a historii celého Vesmíru, zahrnujícího stovky miliard galaxií, z nichž každá sestává ze stovek miliard hvězd [2].

Ale i dnes je nebývalé množství praktických aplikací, bez kterých si moderní život nedovedeme představit, založeno na astronomických poznatcích—například satelitní navigace, komunikace, sledování počasí atd.

Dnešní vědecká astronomie se liší od té historické. Naše Sluneční soustava je téměř zcela zmapována. Krom neustálého dohledu systémů včasného varování před cizími tělesy prolétajícími soustavou, či monitorování Slunce, není automatizace při pozorování Sluneční soustavy příliš zapotřebí. Pozornost se přesunula ke vzdáleným objektům hlubokého vesmíru. Ať už mluvíme o mapování proměnných hvězd, či hledání exoplanet, získávání dat ze vzdálených slabých zdrojů světla vyžaduje delší expozice a velmi přesné sledování rotující oblohy dalekohledem. Ať už se pozorování provádí v obrovské observatoři v Chile, či s malým dalekohledem na zahradě za domem, pohyb všech je třeba řídit podle stejných zákonitostí.

Druhým velkým uplatněním v praxi je astrofotografie. Spíše než o vědu se jedná o formu umění—získávání a úpravu výsledného obrazu velmi vzdálených a tedy i velmi slabých objektů. Od svého zrodu se zájem astrofotografie nikterak nezměnil, objevily se však nové prostředky a možnosti. Fotografie se dnes získávají digitálně, složením řady dlouhých expozic v různých částech světelného spektra. Úžasné fotografie, které vídáme dnes v astronomických časopisech, jsou produktem velice dlouhých akumulovaných expozic a následného náročného zpracování pokročilými matematickými postupy.

2.2 Souřadnicové systémy

Přesná specifikace souřadnicových systémů je nezbytná pro korektní fungování řídicí jednotky. Montáže astronomických dalekohledů pracují se sférickými souřadnicemi. Sférický souřadnicový je určen definicí základní roviny a základního směru. Řídicí jednotka umí pracovat se dvěma při pozorování nejběžněji používanými souřadnicovými systémy—azimutálním a rovníkovým.

2.2.1 Azimutální souřadnicový systém

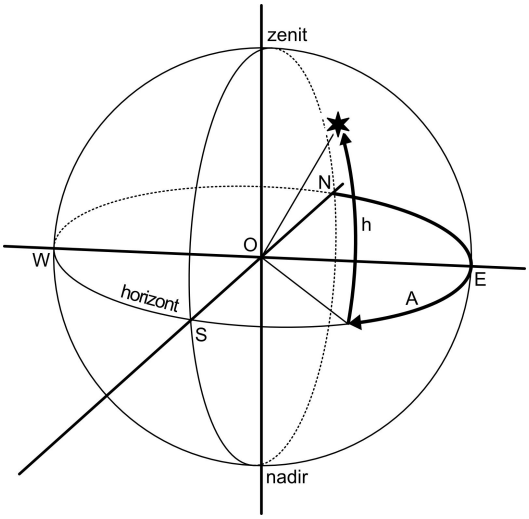
Informace v této podsekci byly převzaty z [9]. Azimutální, nebo také obzorníkový, souřadnicový systém se řadí mezi souřadnice topocentrické—počátek souřadnic se nachází v místě pozorovatele.

Pokud bodem pozorovatele povedeme přímku se směrovým vektorem rovným vektoru tíže, tato přímka protne nebeskou sféru ve dvou bodech—zenitu (nadhlavníku) a nadiru (podnožníku). Kolmá na tuto přímku je základní rovina procházející bodem pozorovatele představující místní horizont. Budeme-li pak vést poledník pozorovacím místem, tzv. místní poledník, protne nám rovinu horizontu ve dvou bodech—v severním bodě N a jižním bodě S.

Dříve astronomický azimutální souřadnicový systém používal za výchozí bod soustavy považoval bod jižní, čímž se lišil od systému občanského. Dnes už se za výchozí bod volí bod severní i v astronomii a tyto dva systémy jsou shodné. Kladný směr ve zvolen doprava od bodu N.

Pokud bychom vedli základním poledníkem rovinu, průnik této roviny s nebeskou sférou by představoval kružnici procházející 4 body—severním, nadhlavníkem, jižním a podnožníkem. Tato kružnice se nazývá místní meridián a představuje místní nebeský poledník.

Poloha objektu na obloze je tedy určena dvěma úhly—azimutem, značeným A, a výškou nad obzorem, značenou h.



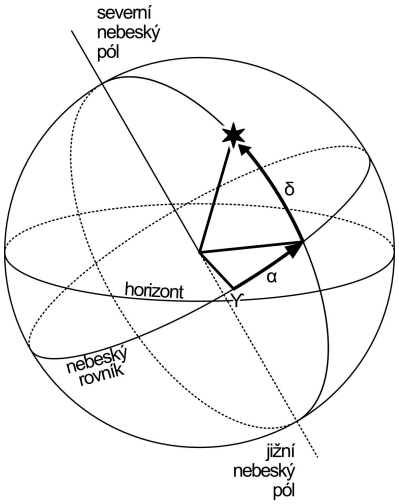
Obrázek 2.1: Ilustrace azimutálního souřadného systému

2.2.2 Ekvatoreální souřadnicový systém

Informace v této podsekci byly převzaty z [9]. Ekvatoreální, také rovníkový, souřadnicový systém vztahuje počátek soustavy ke středu Země, jedná se tedy o geocentrický souřadnicový systém. Hlavní rovinou je rovina zemského rovníku. Základní směr je směr jarního bodu—místa kde se na obloze nachází Slunce v okamžiku jarní rovnodennosti.

Vzhledem ke vzdálenostem pozorovaných objektů je posun mezi polohou pozorovatele a středem Země natolik malý, že jej lze zanedbat. Je tedy možné za počátek soustavy souřadnic brát pozici pozorovatele. Výjimkou je Měsíc, vůči jehož vzdálenosti od Země není poloměr Země zanedbatelný a polohu pozorovatele na povrchu Země je nutno započítat. Odchylka je měřitelná i u dalších těles Sluneční soustavy. Podle požadované přesnosti pak může být nutno tento fakt také zohlednit.

Poloha objektu na obloze je určena dvěma úhly—rektascenzí, značenou α, která určuje odchylku objektu od jarního bodu Υ promítnutou do roviny rovníku, a deklinací, značenou δ, která představuje výšku nad nebo pod rovinou rovníku.



Obrázek 2.2: Ilustrace ekvatoreálního souřadného systému

2.2.3 Porovnání azimutálního a ekvatoreálního systému

Azimutální systém je velmi intuitivní, pro použití v astronomii má ale závažné nedostatky. Jelikož je vztažen k místnímu horizontu, souřadnice objektu na obloze je různá pro pozorovatele na jiných místech Země a mění se také s časem. Mimo to jsou změny souřadnic v čase nerovnoměrné. Azimutálně montovaný astronomický dalekohled musí vykonávat dva nerovnoměrné pohyby při sledování astronomického objektu. Mimo to se zorné pole při sledování stáčí a to také nerovnoměrně.

Azimutálně montované dalekohledy také nemohou sledovat objekt blízko zenitu. Změna azimutu blízko zenitu je velmi rychlá (prochází-li objekt přesně zenitem, je požadovaná změna azimutu teoreticky nekonečně rychlá) a může překračovat mechanické možnosti montáže. Dosáhne-li objekt limitu vzdálenosti od zenitu, musí azimutálně montované dalekohledy přerušit sledování a navázat mohou, až se objekt od zenitu opět vzdálí nad limitní vzdálenost.

Ekvatoreální systém jednoznačně určuje pozici objektu na obloze pro všechny pozorovatele nezávisle na čase. Ekvatoreálně montovaný astronomický dalekohled vykonává při sledování objektu na obloze jeden rovnoměrný pohyb okolo polární osy. Před začátkem pozorování je nutno synchronizovat souřadnicový systém montáže s rovníkovým souřadnicovým systémem.

2.2.4 Převody mezi souřadnými systémy

Většina funkčnosti jednotky si vystačí s ekvatoreálním souřadným systémem. Některé funkce ale vyžadují souřadnice v horizontálním souřadném systému, jehož souřadnice nejsou závislé na čase (viz 3.5.7). Ve své knize *Astronomical Agorithms* uvádí Meeus následující rovnice pro převody mezi souřadnými systémy [5].

$$\begin{aligned} \tan A &= \frac{\sin H}{\cos H \sin \varphi - \tan \delta \cos \varphi} \\ \sin h &= \sin \varphi \sin \delta + \cos \varphi \cos \delta \cos H \end{aligned} \tag{2.1}$$

Rovnice 2.1 popisuje výpočet azimutálních souřadnic z ekvatoreálních souřadnic místa na obloze a geografických souřadnic pozorovatele. V rovnici A značí azimut místních azimutálních souřadnic a h je výška nad obzorem. Řeckým písmenem φ je značena zeměpisná šířka pozorovatele a řecké písmeno δ označuje deklinaci. Rektascenze i zeměpisná dálka jsou obsaženy v tzv. místním hodinovém úhlu, značeném H.

Místní hodinový úhel představuje místní hvězdný čas upravený o rektascenzi. Přitom místní hvězdný čas lze získat z greenwichského hvězdného času upraveného o zeměpisnou délku. Pokud θ představuje místní hvězdný čas, θ_0 greenwichský hvězdný čas a L pozorovatelovu zeměpisnou šířku (kladná k západu), pak místní hodinový úhel lze vypočítat jako

$$H = \theta - \alpha \tag{2.2}$$

nebo

$$H = \theta_0 - L - \alpha \tag{2.3}$$

Pro převod místních azimutálních souřadnic na souřadnice ekvatoreální lze použít rovnici 2.4, která je úpravou předchozí rovnice pro převod ekvatoreálních souřadnic na souřadnice azimutální.

$$\begin{aligned} \tan H &= \frac{\sin A}{\cos A \sin \varphi + \tan h \cos \varphi} \\ \sin \delta &= \sin \varphi \sin h - \cos \varphi \cos h \cos A \end{aligned} \tag{2.4}$$

Výsledkem rovnic 2.4 je deklinace a hodinový úhel. Rektascenze je získána přepočtem z hodinového úhly jednoduchým vyčíslením z rovnic 2.2 nebo 2.3.

2.3 Časy a data v astronomii

Astronomické objekty jsou často pozorovány po delší časové úseky, které nezřídka dosahují délky i mnoha měsíců, z mnoha míst na Zemi. Je proto důležité jednoznačně určit používané časy, aby bylo možné tato pozorování synchronizovat.

2.3.1 Hvězdný čas

Hvězdný čas je hodinový úhel jarního bodu vztažený k meridiánu. V okamžiku svrchního průchodu jarního bodu meridiánem je 0h0m0s hvězdného času. Hodinový úhel 15° odpovídá hvězdnému času jedné hodiny. Je užitečné poznamenat, že při západu jarního bodu je 6 hodin hvězdného času. Hvězdný čas proto neslouží k měření časových intervalů, ale indikuje, kdy daný objekt vrcholí na noční obloze.

2.3.2 Světový čas

Světový čas, je aktuální čas na nultém poledníku. Bývá označován jako Coordinated Universal Time, krátce UTC.

2.3.3 Lokální čas

Lokální čas je světový čas upravený o hodnotu danou časovým pásmem. Časová pásma skokově upravují světový čas o +-12 hodin. Rozdělení časových pásem vychází ze zeměpisné délky, definitivně určeno je však domluvenou konvencí.

2.3.4 Juliánské datum

Číslo juliánského dne je hodnota jednoznačně určující počet dní uplynulých od počátku Juliánské periody. Počátek první Juliánské perioda byl stanoven na 1. ledna 4713 př. n. l.

Juliánské datum kteréhokoli okamžiku je pak číslo juliánského dne plus zlomek uplynulého dne od posledního poledne světového času. Juliánské datum tak přesně určuje nejen den, ale i čas jediným číslem a proto se používá v astronomii téměř výhradně.

Změna juliánského dne v poledne na místo půlnoci je pro běžný život neobvyklá, pro určování časů astronomických pozorování je ale výhodná, protože celé pozorování v průběhu jedné noci proběhne v rámci stejného juliánského dne. Samozřejmě tato výhoda se týká téměř výhradně Evropy, kde odchylka lokálního a světového času není příliš velká.

2.4 Montáže dalekohledů

2.4.1 Azimutální

Azimutální montáže mají první osu kolmou na horizontální rovinu. Tato osa se nazývá azimutální osa a nabývá hodnot od 0° do 360°. Druhou osou je výšková osa nebo také elevace. Ta nabývá hodnot -90° až 90°. Obecně platí, že viditelné hvězdné objekty mají tuto souřadnici větší než 0°. Za podmínek, kdy obrys okolní krajiny však spadá pod rovinu horizontu, může mít pozorovaný objekt tuto souřadnici vlivem deprese horizontu i zápornou.

2.4.2 Rovníkové

Toto upevnění dalekohledu má první osou rovnoběžnou s osou rotace Země. Tato osa se nazývá rektascenční osa. Druhá osa je kolmá na první osu a nazývá se deklinační osa. Deklinace představuje odchylku objektu na nebeské sféře od průmětu zemského rovníku. Podle konstrukce se rovníkové montáže dále dělí na vidlicové a německé.

Vidlicové (Anglické)

Tubus dalekohledu je uchycen ve vidlici. Tento typ dokáže sledovat objekt přes meridián. Vidlice jsou však náročné na výrobu a má-li být zachována dostatečná tuhost, musí být vidlice robustní a těžká. Vidlice také omezuje délku zadní části tubusu. Dalekohled umístěný ve vidlici je obtížné vyvážit při změně vybavení uchyceného na tubusu.

Obrázek 2.3: Vidlicová montáž dalekohledu městské hvězdárny ve Zlíně

Německé (GEM—German Equatorial Mount)

Německé montáže jsou jednoduché konstrukce, snadno se vyvažují díky posuvnému protizávaží. Pro dalekohled je podstatná poloha od pilíře—pokud je tubus východně od pilíře, lze namířit na západní polosféru a naopak. Sledování objektu přes meridián je tedy zpravidla možné jen do určitého mezního úhlu a po jeho překročení je potřeba dalekohled přeložit na druhou stranu od pilíře.

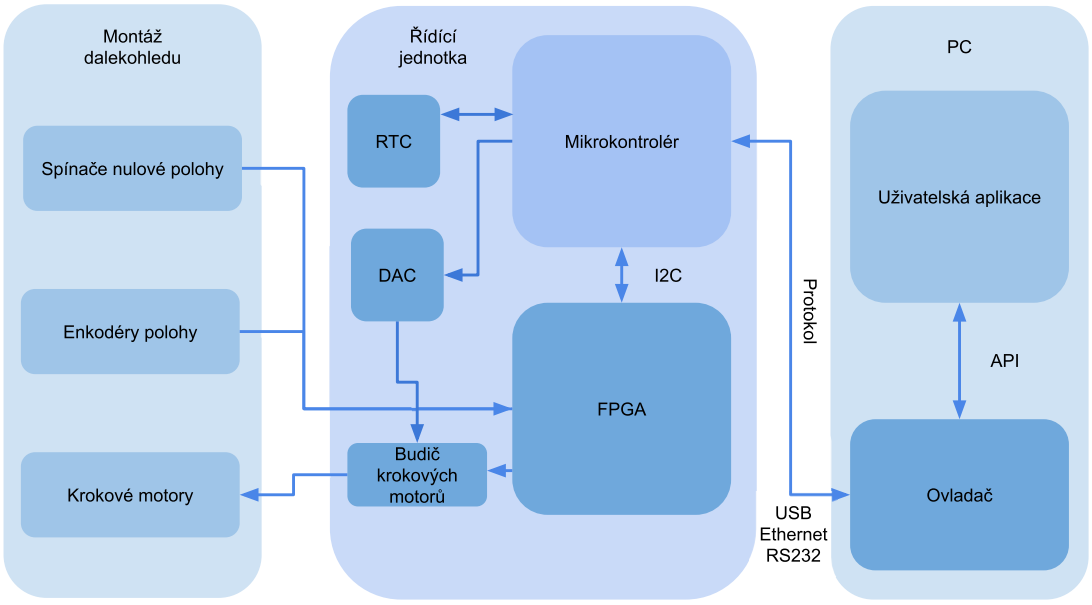
Obrázek 2.4: Německá montáž dalekohledu na hvězdárně BS Observatory

Kapitola 3

Architektura systému

Jednotka ovládání montáže dalekohledu je postavena na mikrokontroléru a programovatelném hradlovém poli. Přesné řízení krokových motorů s odchylkou menší než oblouková vteřina za 24 hodin provozu je těžko realizovatelné v mikrokontroléru, pokud má zastávat i další funkce, jako například komunikaci, správu stavu a matematické výpočty. Jako řešení se nabízí použití paralelního mikrokontroléru jako hard real-time systému. Pokud ale řešení vyžaduje další mikročip na desce, není důvod rovnou nepoužít programovatelné hradlové pole. To je optimální pro řešení daného problému—přesné časování dané hodinovým signálem, kompletní paralelnost (komunikace nijak nezatěžuje zbytek čipu) a přímé a přesně časované ovládání výstupních signálů bez nutnosti použití nějakého GPIO řadiče.

3.1 Schéma implementace



Obrázek 3.1: Blokové schéma jednotky, viz 3.1

3.2 Hardware

Tabulka 3.1: Přehled hardware komponent jednotky

MCU	Atmel 32UC3A1512-U
FPGA	Lattice LCMXO2-2000HC
Řadiče krokových motorů	OnSemiconductor LV8729
Obvod reálného času	Epson RX8900CE
Číslicově-analogový převodník	Texas Instruments TLV5620

Jelikož jednotka není čistě záležitostí software a její účel je připojení k fyzické montáži dalekohledu, hardware jednotky výrazným způsobem ovlivňuje její použitelnost—ať už se bavíme o rozhraní, přes která je možné jednotku připojit k periferiím a ovládacím zařízením—až po fyzikální limity součástek používaných v jednotce.

3.2.1 Rozhraní

Podstatou jednotky je zpracovávat příkazy z připojeného řídicí zařízení (PC, ruční ovládání) a propagovat tyto informace dále to motorů montáže. Jako taková musí mít jednotka dostatečné technické vybavení pro připojení všech potřebných periferií.

Základem jsou konektory potřebné pro připojení řídicích zařízení. Všechna tato rozhraní jsou napojena na MCU. Z pohledu práce je podstatné propojení s PC přes sériovou linku. K tomu jsou k dispozici dva standardní RJ25 konektory. Jednotka je schopna přijímat a zpracovávat sériovou komunikaci na obou rozhraních zároveň. Dva konektory jsou na desce přítomny z důvodu možnosti současného připojení řídicího PC i jednotky ručního ovládání. Jednotka ručního ovládání komunikuje stejným komunikačním protokolem a vůbec se neliší od počítače.

Rozhraní pro komunikaci s montáží je poněkud rozmanitější. Jednotka musí spravovat řadu rozličných funkcí, které jsou téměř všechny napojeny přímo na GPIO piny FPGA. Jedinou výjimkou jsou konektory pro připojení krokových motorů, které jsou zapojeny na výstupy řadičů krokových motorů. Logické signály FPGA jsou zpracovány řadičem, který podle těchto signálů nastavuje proud na dvou párech vodičů napojených na cívky motorů. Mimo to jednotka obsahuje dva výstupní konektory přímo s logickými signály kroku, směru a mikrokroků, pro připojení na externí řadič motorů. To je vhodné v případě, když použité motory vyžadují větší proud, než který je schopen integrovaný řadič dodávat. Pak může být využita veškerá funkčnost jednotky a pro ovládání krokových motorů je připojen pouze koncový člen (budič) s vyšším proudovým rozsahem, než dovoluje obvod integrovaný na desce plošných spojů jednotky. Dále jednotka obsahuje konektor vstupu enkodérů a spínačů nulové polohy. Tyto rozšíření montáže je pouze volitelné a jednotka je pro správný chod nepotřebuje. Posledním vstupním konektorem je RJ25 konektor pro připojení kamery pro automatickou korekci obrazu.

3.2.2 Programovatelné hradlové pole

Časově kritické funkce a funkce, jejichž výstupem jsou logické signály využívané ostatními komponentami jednotky jsou realizovány v programovatelném hradlovém poli 3.1 (zkráceně FPGA z anglického *Field-programmable Gate Array*). Pouzdro obvodu FPGA je standardní TQFP100. Funkce, které FPGA zastává jsou detailně popsány v kapitole 4.

FPGA je propojené s MCU přes I2C sběrnici, která je hlavním prostředkem komunikace a předávání příkazů. Mimo to deska obsahuje několik propojení GPIO pinů mezi MCU a FPGA, které umožňují rychlé předání jednobitových informací.

3.2.3 MCU

Veškeré výpočetně náročné funkce jako např. funkce sestávající z většího množství sekvenčních kroků, ale také komunikace s řídicími zařízeními, správa konfigurace, ovládání FPGA, komunikace s RTC a ovládání DAC provádí mikrokontrolér 3.1 (zkráceně MCU z anglického *Microcontroller Unit*). Pouzdro obvodu FPGA je standardní TQFP100. Funkce, které MCU zastává, jsou detailně popsány v kapitole 5.

3.2.4 Řadič krokových motorů

Jednotka obsahuje dva řadiče krokových motorů 3.1, jeden pro každou osu montáže. Obvod implementuje proudem ovládané, pulzně-šířkově regulační řadiče dvoupólových motorů.

Pulzně-šířkové modulace se využívá v mechanismu mikrokroků, který jednotka podporuje a který je klíčový pro aplikaci v řízení montáže. Mikrokrokování umožňuje postupně měnit proud na cívkách, čímž je přepólování dosaženo ne v jednom kroku, ale v nastaveném počtu mikrokroků. Počet mikrokroků je nastavován jednou tříbitovou hodnotou n a jeho hodnota je poté rovna

$$2^n, \text{ kde } n \in [0..7]$$

Při změně počtu mikrokroků vynuluje řadič čítač mikrokroků a nastaví proudy do cívek na výchozí hodnoty definované ve specifikaci. Pokud změna proběhne před dokročením do celého kroku, řadič do tohoto kroku skočí. To v lepším případě způsobí škubnutí, v tom horším přeskočení kroků, či poškození mechaniky montáže.

Oproti změně počtu mikrokroků, změna směru pouze zamění odčítání a přičítání u interního čítače mikrokroků. Řadič přejde do předchozího stavu proudu na cívkách namísto následujícího a pokračuje opačným směrem.

Důležitým vstupem řadiče je analogový pin referenčního napětí. Napětí na tomto pinu určuje celkový proud, který řadič dává do motorů. Proud ovlivňuje rychlost změny a intenzitu magnetického pole generovaného cívkami. Větší proud dovoluje vyšší rychlosti a také znamená silnější sevření motoru—je těžší vnější silou přeskočit krok. Napětí na tomto analogovém pinu je nastavované samostatným DAC na desce, který je ovládán mikrokontrolérem. Při rychlejších pohybech se proud dynamicky nastavuje větší, než při pomalém sledování. Proud do motorů je určen rovnicí

$$I_{OUT} = \frac{\frac{V_{REF}}{5}}{R_{F1}} \tag{3.1}$$

kde referenční napětí V_{REF} je výstupem digitálně-analogového převodníku 3.2.6 a R_{F1} na desce má odpor $0,22\,\Omega$.

Posledním, pro jednotku podstatným signálem řadiče je signál kroku. Vždy na náběžnou hranu tohoto signálu řadič přejde do následujícího stavu mikrokroku.

3.2.5 Obvod hodin reálného času

Pro potřeby astronomických výpočtů je na desce přítomen obvod reálného času 3.1. Obvod je nezávislý na vnějším napájení díky dedikované baterii na desce. To umožňuje RTC

zachovávat svůj stav mezi cykly zapnutí a vypnutí jednotky. Obvod komunikuje s mikrokontrolérem přes dvě rozhraní. První, jednodušší rozhraní je přímé propojení výstupního signálu přerušení na GPIO piny mikrokontroléru. To umožňuje mikrokontroléru využít přerušení v pravidelných a nastavitelných časových intervalech.

Hlavním způsobem komunikace RTC modulu je ale I2C sběrnice, která dovoluje mikrokontroléru číst a zapisovat do registrů RTC. Vedle konfiguračních registrů jsou v RTC k dispozici také registry aktuálního času a data pro čtení a zápis. To umožňuje jednotce zjistit informaci o aktuálním datu a také toto datum na základě příkazu z připojeného PC přepsat.

3.2.6 Digitálně-analogový převodník

Na desce jednotky je přítomen digitálně-analogový převodník 3.1 pro generování referenčního napětí na vstupu řadičů krokových motorů. Řadiče na základě tohoto napětí mění proud vedený do cívek krokových motorů a pro potřeby jednotky je podstatné mít možnost tento proud měnit. DAC je napojený na GPIO výstupy mikrokontroléru, který má kontrolu nad výstupním napětím.

Výstupem obvodu převodníku jsou čtyři piny. Na těchto pinech nastavuje převodník napětí podle hodnot zapsaných do interních registrů. Zápis je řízen třemi signály na vstupu DAC. Primárním signálem je signál `LOAD`. Pokud je tento signál nastaven na log. 1, převodník čte vstupní hodnotu na signálu `DATA` do jedenácti-bitového posuvného registru vždy na změnu signálu `CLK` z log. 0 na log. 1. Při změně signálu `LOAD` zpět na log. 0 zpracuje převodník data v posuvném registru. Dva první bity určují jeden ze čtyř výstupů, jehož hodnota má být zápisem aktualizována. Třetí bit je tzv. *RNG* bit, který určuje zesílení výstupu. Zbylých osm bitů nahraných do posuvného registru jako posledních určují zlomek vstupního referenčního napětí, který bude nastaven na odpovídajícím výstupním pinu. To je určeno vzorcem

$$V_O = REF \times \frac{CODE}{256} \times (1 + RNG) \tag{3.2}$$

kde REF je referenční napětí (kterému na desce jednotky odpovídá napětí $3V$), $CODE$ je hodnota určená dolními osmi bity a RNG je bit zesílení.

3.3 Inter-integrated Circuit (I2C) sběrnice

I2C je sběrnice navržená společností Philips Semiconductor, dnes známé jako NXP Semiconductors, pro jednoduchou komunikaci mezi zařízeními na plošném spoji. Jde o synchronní, sériovou, polo-duplexní, multi-master multi-slave sběrnici využívající pouze dva vodiče pro komunikaci—SCL, hodinový signál, a SDA, datový vodič [7]. Komunikace využívající pouze dva vodiče vyžaduje protokol, jehož implementace je popsána v kapitole 4.4.1. Jelikož je celá jednotka z pohledu I2C navržena jako single-master multi-slave, problémy s rozdělováním času na sběrnici v řešení odpadají. V návrhu jednotky je mikrokontrolér master zařízením. Ten využívá sběrnice pro komunikaci s FPGA a RTC obvodem tam, kde jednoduché signály nestačí pro přenos informace.

3.4 Komunikace s PC

3.4.1 Aplikační rozhraní

Aplikační rozhraní (zkrácené API, z anglického *application programming interface*) ovladače se řídí specifikací API programu *Scientific Image Processing System* společnosti Moravské přístroje, který je používán ke získávání a zpracování astronomických snímků pro výzkum a astrofotografii.

Z důvodů zpětné kompatibility nové verze pouze rozšiřují stávající API novými, volitelnými příkazy, které nemusí připojená jednotka, ani software podporovat. Veškeré využití API je ovšem omezeno na aplikaci a ovladač běžící na připojeném PC. Jednotka je od API zcela odstíněna pomocí mezivrstvy, kterou tvoří ovladač. Ten se stará o namapování funkcí exportovaných v API na korektní příkazy protokolu. Ovladač není součástí této práce.

3.4.2 Komunikační protokol

Komunikační protokol ovladače je navržen a optimalizován pro potřeby jednotky. Protokol je implementován dedikovaným ovladačem, který vytváří mezivrstvu mezi aplikačním rozhraním využívaným v uživatelských aplikacích a komunikačním protokolem využívaným jednotkou.

Data příkazů protokolu jsou poslána přímo rozhraním USB či RS232C, nebo jsou zapouzdřena do Ethernet paketu. Samotná struktura protokolu se nemění s různými způsoby komunikace.

Komunikační protokol je ve formátu prostého textu. Tento přístup proti např. binárnímu protokolu byl zvolen pro usnadnění ladění. Mírný nárůst objemu přenášených dat ve srovnání s jinými, úspornějšími typy protokolu je vzhledem k rychlosti převážně používaných rozhraní (USB, Ethernet) zanedbatelný, ale ani u sériové komunikace nepředstavuje obtíž.

Protokol je založený na výměně textových řetězců. Začátek řetězce je identifikován znakem `$`. Při sériové komunikaci jednotka očekává tento znak a zahazuje příchozí znaky, dokud neodpovídají počátečnímu znaku. Při komunikaci přes ostatní rozhraní, kde je příchozí celý řetězec jednotka zahodí všechny znaky z řetězce, dokud nenarazí na znak `$`. Následně jsou u sériové komunikace načítány znaky, dokud jednotka nedostane ukončovací znak. Ten je definován jako `^`. U komunikace přes ostatní rozhraní probíhá procházení řetězce do doby, než jednotka nenarazí na ukončovací znak. Při jeho nalezení překopíruje rozsah mezi nalezeným počátečním a koncovým znakem do příchozího bufferu.

Délka žádného příkazu komunikačního protokolu nepřekračuje limit 64 znaků (64 bajtů). To dovoluje použít při přenosu po USB základní a vždy podporovanou délku tzv. "bulk" paketu 64B. I při sériové komunikaci je testováno, je-li ukončovací znak nalezen do počtu 64 znaků a pokud ne, jednotka příchozí buffer odmítne.

Kompletní seznam všech příkazů komunikačního protokolu je uveden v tabulce A.1. Začátek příkazu posílaného připojeným zařízením je vždy znak `$`. Odpovědi posílané jednotkou začínají buď tečkou, nebo znakem `!`. Tečka v odpovědi značí pozitivní odpověď, zatímco `!` indikuje chybu. Každý řetězec odpovědi je také je ukončen znakem `^`. V každém příkazu je protokolu je obsažen unikátní tří-znakový kód. Tento kód se také používá pro velkou část interní identifikace ve zdrojovém kódu, jako jsou jména proměnných, typů a funkcí.

3.4.3 Konfigurační příkazy

Konfigurační příkazy se od ostatních příkazů odlišují v tom, kdy a za jakých okolností dochází k jejich zpracování. Přestože jsou součástí komunikačního protokolu, ovladač jednotky tyto příkazy z nepodporuje. Jinými slovy—konfigurace jednotky není podle návrhu podporována standardními aplikacemi. Pro nastavení pomocí konfiguračních příkazů je zapotřebí speciální dedikované aplikace.

Některá konfigurační data jsou pro fungování jednotky nezbytně důležitá. Bez informací, jako je například počet kroků na otáčku jednotlivých os motorů nedává použití jednotky smysl. V ten moment nelze provádět pohyb na souřadnice či sledování oblohy. Z tohoto důvodu jednotka nereaguje na ostatní příkazy protokolu, dokud není zcela nakonfigurována. Tato konfigurace může přijít z konfigurační aplikace nebo může být načtena z FLASH paměti jednotky.

Kompletní popis konfiguračního protokolu je uveden v tabulce A.3. Formát obsahu sloupců je zcela shodný s formátem obsahu tabulky A.1.

3.4.4 Sériová komunikace (USART)

Sériová komunikace v jednotce je implementována s využitím knihovny dostupné v rámci Advanced Software Framework 4 [6]. Tato knihovna vyváží funkce pracující nad registry USART rozhraní, které jsou namapované na paměťový prostor mikrokontroléru. To dovoluje jednoduchý zápis i čtení znaků po sériové lince.

Čtení je implementováno pomocí přerušení s nízkou prioritou, ve kterých dochází k vyčtení znaku z bufferu rozhraní. Zápis funguje podobným způsobem, ovšem použití přerušení je skryto v implementaci knihovních funkcí a není dokumentováno. Při zápisu znaku je přerušení vyvoláno po odeslání znaku elektronikou a dovolí funkci zápisu opustit smyčku. Vektor přerušení je nastaven ve funkci, která inicializuje USART rozhraní. To může způsobit potenciální problémy při náročných časových požadavcích na mikrokontrolér, tato situace v této implementaci nenastává.

3.5 Funkce řídicí jednotky

Tato kapitola se zabývá teoretickým popisem funkcí. Nerozvádí konkrétní způsoby implementace, jako spíše obecný mechanismus a myšlenku za řešením problematiky.

3.5.1 Synchronizace s nebeskou sférou

Synchronizace je jednou z nejpodstatnějších funkcí pro fungování celé jednotky. Aby mohla jednotka provádět jakékoliv automatizované operace, musí si uchovávat přehled o souřadnicích, na které míří a ke kterým vztáhnout například souřadnice přijaté příkazem přesunu na souřadnice.

Tento úkol je v jednotce realizován následovně. Jednotka má vždy absolutní představu o pozicích krokových motorů, tedy o absolutním stavu montáže, díky přesnosti FPGA, které nemůže vynechat krok. Tuto informaci je ale potřeba vztáhnout k souřadnicím na nebeské sféře.

Pro tuto operaci potřebuje jednotka následující informace:

1. referenční bod, na který dalekohled míří v rovníkových souřadnicích
2. absolutní pozice os montáže—čítačů kroků—kterým tyto souřadnice odpovídají

3. čas ve který je tento vztah mezi nebeskými souřadnicemi a absolutními pozicemi platný

Referenční bod je jednotce vždy předán jako parametr příkazu synchronizace a je následně uložen do stavové informace jednotky.

Absolutní pozice os montáže je potřeba namapovat na aktuálně přijatý referenční bod. Místo čtení aktuálních hodnot čítačů jsou tyto čítače pro jednoduchost vynulovány. Od tohoto momentu se předpokládá, že hodnoty 0, 0 odpovídají referenčním souřadnicím v čase přijetí.

Čas okamžiku synchronizace je přečten z RTC modulu na desce jednotky a je uložen do stavových dat spolu s referenčními rovníkovými souřadnicemi. Tento absolutní čas, uložený jako juliánské datum, je potřeba pouze pro zotavení z vypnutí. Pro veškeré výpočty je důležitý pouze čas uplynulý od okamžiku synchronizace. Počet sekund uběhlých od synchronizace si jednotka udržuje za pomocí sekundového přerušení z RTC modulu.

3.5.2 Sledování pohybu astronomických objektů

Nejzákladnější a přitom nejdůležitější funkcí montáže je automatické sledování objektů na hvězdné obloze. S rotací Země se objekty na obloze s časem posunují relativně vůči pozorovateli. Ruční posun objektů zpět do zorného pole při pozorování okem nepředstavuje problém, ovšem při pořizovaní snímků kamerou s delší expozicí není dostatečně přesná, nemluvě o tom, že expozice mohou trvat i desítek minut a takových snímků může být během pozorování pořízeno mnoho desítek.

Prvním krokem k automatizaci montáže se stal rovnoměrný pohyb v záporném směru rektascenze. Rychlost tohoto pohybu závisí na pozorovaném objektu—odvíjí se od doby mezi dvěma následujícími průchody pozorovaného tělesa místním poledníkem. Člověku nejbližší doba oběhu je ta Slunce. V astronomii se nazývá sluneční den. Trvá přesně 24 hodin a je odvozena od doby rotace Země okolo své osy upravené o zlomek doby oběhu Země okolo Slunce. Nejpodstatnější pro astronomii je ale tzv. siderický, neboli hvězdný den, jelikož je od něj odvozena sledovací rychlost všech objektů mimo Sluneční soustavu. Hvězdný den je závislý zcela na době rotace Země okolo své osy—díky vzdálenostem dalekých vesmírných objektů lze veškeré odchylky způsobené pohybem Země okolo Slunce zanedbat. Posledním dnem běžně používaným v astronomii je tzv. měsíční den. Ten, podobně jako sluneční, je odvozen od doby rotace Země okolo své osy, ale upravené o zlomek doby oběhu Měsíce okolo Země. Délku trvání jednotlivých dnů shrnuje následující tabulka 3.2.

Tabulka 3.2: Délky dnů využívaných v astronomii

Hvězdný den	86164,09054 s
Měsíční den	89416,30000 s
Sluneční den	86400,00000 s

Známe-li dobu, za kterou projde sledovaný objekt meridiánem od posledního průchodu, můžeme snadno vypočítat potřebnou sledovací rychlost. Rozměr rychlostí používaných jednotkou je počet kroků motoru za sekundu. Ten zohledňuje úhlovou rychlost objektu a také převod konkrétní mechaniky montáže spolu s rozlišením krokového motoru. Výpočet je poté velmi jednoduchý

$$SPS = \frac{n_r}{t_d} \tag{3.3}$$

kde SPS je počet kroků za sekundu, n_r představuje rozlišení osy montáže, tedy rozlišení motoru $\times$ převod a t_d představuje dobu průchodu sledovaného objektu meridiánem od posledního průchodu. Pro tři doby průchodu tak získáme tři rychlosti:

- *Hvězdná rychlost*—rychlost používána pro sledování hvězd a jiných objektů mimo Sluneční soustavu.
- *Sluneční rychlost*—rychlost používána při sledování Slunce, využívána například při monitorování sluneční aktivity.
- *Měsíční rychlost*—rychlost používána při sledování Měsíce. Dnes nachází využití už téměř výhradně v astrofotografii.

3.5.3 Rampování

Rampování není samostatnou funkcí, kterou by počítač zadával jednotce, jako např. synchronizace či volný pohyb (příkaz `move`). Přesto je schopnost jednotky rampovat zcela nezbytnou součástí. Dalekohled umístěný na montáži, ale také montáž samotná má nezanedbatelnou hmotnost a nelze měnit rychlosti v nulovém čase. Při rychlejších pohybech je nutné rychlost lineárně zvětšovat či opět zmenšovat, aby nedošlo k přeskočení kroků motoru, či poškození montáže.

3.5.4 Volný pohyb

Tento druh pohybu jednotka provádí při operacích, jako je prostý pohyb zadaný počítačem nebo ručním ovladačem.

Při zadání příkazu pohybu v daném směru jednotka tento pohyb započne. V tomto stavu setrvá, dokud nedostane nový příkaz. Pohyb po předem nespecifikovanou dobu je pohyb s nejvyšší prioritou. Uživatel jej zadává neočekávaně a jednotka na něj vždy zareaguje odpovídajícím způsobem.

Pokud je aktivní pohyb najíždění na souřadnice, je zcela přerušen a jednotka pokračuje volným pohybem. Po ukončení volného pohybu jednotka v pohybu na souřadnice nepokračuje. Pohyb na souřadnice je třeba zadat znovu, přičemž v ten moment se propočítá s novými souřadnicemi.

Je-li aktivní sledování, či korekce, jednotka započne volný pohyb podle příkazu. Po skončení volného pohybu se opět vrátí do původního stavu sledování nebo automatické/pulzní korekce, pokud tato nebyla v průběhu pohybu deaktivována.

3.5.5 Pohyb při najíždění na zadané souřadnice

Tento druh pohybu se provádí výhradně při tzv. příkazu `goto`. Oproti volnému pohybu není tento pohyb zadáván jako příkaz aktivace ve směru, ale jako cílové souřadnice. Z aktuálních souřadnice (viz 3.5.1) a získaných souřadnic je vypočítán úhel, který se s využitím současného nastavení počtu kroků na otáčku každé osy přepočítá na počet kroků. Tento počet kroků je následně předán do FPGA a je aktivován přesun na souřadnice.

Pokud je aktivní volný pohyb, jednotka neprovede žádnou akci. Tato kontrola by měla být provedena před výpočtem souřadnic a zápisem do FPGA pro ušetření času, neboť astronomické výpočty jsou poměrně náročným výpočetním úkonem.

Je-li aktivní sledování, či korekce, jednotka započne pohyb na souřadnice podle příkazu. Po skončení pohybu se opět vrátí do původního stavu sledování nebo automatické/pulzní korekce, pokud tato nebyla v průběhu pohybu deaktivována.

3.5.6 Přeložení tubusu dalekohledu

Popis této funkce je založen na konceptu návrhu a není implementován v současném řešení.

Překládání dalekohledu na druhou stranu od pilíře je základní funkčností pro použití jednotky s německým typem montáží. Tento typ montáže dovoluje sledovat objekty na hvězdné obloze pouze určitý úhel za meridián. Přesná hodnota tohoto úhlu závisí na konstrukci konkrétní montáže. Od určitého okamžiku je ale vždy nutné montáž přeložit.

Přeložení dalekohledu je pohybová operace, u které nedochází ke změně souřadnic, na které dalekohled míří. Po dokončení přeložení míří dalekohled na stejné rovníkové souřadnice na obloze, ale tubus se nachází na opačné straně od pilíře, což montáži dovoluje pokračovat ve sledování objektu na druhé svislé polo-sféře. Přeložením tubusu dojde k otočení zorného pole o 180 stupňů, ale tuto skutečnost montáž nemůže ovlivnit a musí se s ní vypořádat software ovládající kameru nebo zpracovávající získaná data.

Pokud k přeložení dochází v momentu, kdy osa dalekohledu prochází rovinou meridiánu, pak není překlopení nic většího, než pohyb o polovinu celé otáčky v obou osách. Při přeložení tam a zpět je třeba brát v potaz kabely připojené k montáži. Překládání zpět musí probíhat v opačném směru pohybu, než překládání tam, tedy nesmí dokončit celou otáčku. Obecné přeložení pak zahrnuje odchylku aktuálních souřadnic od roviny meridiánu a její kompenzaci pří překládání.

3.5.7 Parkování

Popis této funkce je založen na konceptu návrhu a není implementován v současném řešení.

Pro zvýšení pohodlí a usnadnění používání obsahuje jednotka funkci tzv. parkování. To spočívá v zapamatování aktuálních rovníkových souřadnic a aktuálního času v okamžiku přijetí příkazu a následné najetí na horizontální souřadnice uložené v paměti. To umožňuje dalekohled nasměrovat do pozice, ve které lze jednotku bezpečně vypnout a po zapnutí, pokud s dalekohledem nebylo v době vypnutí pohnuto, odparkování uvede dalekohled opět do stavu synchronizovaného s nebeskou sférou.

Samotný pohyb parkování je realizován obdobně, jako pohyb `goto`, s tím rozdílem, že cílové souřadnice, zadány v horizontálním systému, je potřeba nejdříve přepočítat na rovníkové. Poté se z rozdílu souřadnic vypočítají úhly a z úhlů počty kroků.

Dokončení parkování vypne sledování i ostatní druhy pohybu. Jednotka však dále reaguje na příkazy a je možno ji kterýmkoli příkazem opět rozpohybovat. Pro možnost korektního odparkování po takovémto pohybu je ale nutné znovu zaslat příkaz parkování. Alternativně je možné místo opětovného zaparkování po pohybu parkovací souřadnice aktualizovat. Souřadnice na které má jednotka parkovat nejsou zasílány protokolem, ale vyčtou se z aktuálních souřadnic příkazem `set_park`. Proto je užitečné mít možnost parkovací pozici po zaparkování ručně upravit a následně aktualizovat.

3.5.8 Pulzní korekce

Pulzní korekce pozice dalekohledu dovoluje udržovat pozici sledovaného objektu na základě vyhodnocení polohy hvězdy snímané druhou, tzv. "pointační" kamerou. Tyto korekce, typicky prováděné v intervalech několik jednotek až desítek sekund, dovolují kompenzovat např. mechanické nepřesnosti a chyby v převodech (např. periodickou chybu šnekových převodů), mechanické deformace tubusu a pozice optických prvků dalekohledy apod.

Historicky jsou pulzní korekce řešeny pomocí samostatného rozhraní, obsahujícího čtyři signály pro pohyb oběma směry v obou osách. Použitá řídicí jednotka je tímto rozhraním vybavena a dokáže korigovat pohyb na základě těchto vstupů. Ovšem použití fyzického rozhraní vyžaduje zdroj řídicích signálů.

3.5.9 Automatické ukládání konfigurace

Popis této funkce je založen na konceptu návrhu a není implementován v současném řešení.

Jednotka obsahuje velké množství konfiguračních i stavových informací, z nichž je velkou část potřeba zachovat. Ne všechny informace jsou dostupné mikrokontroleru, který spravuje paměť a je potřeba je nejprve přečíst. Z důvodu zachování těchto informací (jako je například stav sledování) mezi stavy vypnutí je potřeba tyto informace uložit.

Jednotka je nastavena, aby čekala vždy 5 sekund po přijetí posledního příkazu a poté všechny informace uložila. Pokud následně jednotka dostane další příkaz, časovač se vynuluje a po dalších 5 sekundách nečinnosti se uložení provede opětovně. Tento přístup byl zvolen jako kompromis mezi pohodlím uživatele a šetřením zápisů do interní FLASH paměti mikrokontroléru. Pětisekundový interval nečinnosti nastane zpravidla pouze po vypnutí či odpojení ovládajícího programu, který jinak podle specifikace zasílá požadavek na stav jednotky každou sekundu.

Kapitola 4

Návrh HW pro FPGA

Původní návrh řídicí jednotky obsahoval programovatelné hradlové pole (FPGA) Lattice LCMXO2-1200HC. Tato verze FPGA obsahuje 1280 look-up tabulek, což se podle počátečních odhadů zdálo pro implementaci dostačující. Během vývoje se ale ukázalo, že tento počet neumožní implementaci kompletní funkcionality. Následná změna implementace rampovací tabulky s použitím Embedded Block RAM (EBR) dále zvedla požadavky na FPGA. Varianta 1200HC obsahuje pouze 7 bloků EBR a to limituje velikost jedné tabulky na 768 32-bitových položek, která tak zabere 3 EBR bloky. Pro dvě osy zůstane jeden ERB blok nevyužit.

FPGA čip Lattice LCMXO2-2000HC je dostupný ve stejném 100-pinovém TQFP pouzdře jako verze 1200HC, což umožňuje jeho použití bez nutnosti jakýchkoliv úprav na desce plošného spoje. 8 EBR bloků dovoluje realizovat dvě tabulky o 1024 32-bitových položkách pro tabulku rychlostí. To šetří logiku nutnou pro realizování výpočtu mikrokroků. Hodnoty a popis hardware vybavení FPGA je převzat z dokumentace MachXO2 rodiny FPGA [3].

4.1 Návrh interní logiky

FPGA je zodpovědné za ovládání krokových motorů a jejich automatický chod podle daného nastavení, vyčítání informací z enkodérů polohy a indikátorů nulové polohy. Pro komunikaci přes I2C je z pohledu mikrokontroléru v FPGA k dispozici paměťový prostor. To umožňuje MCU číst i zapisovat data dle vlastní potřeby bez nutnosti FPGA tyto operace jakkoli potvrzovat.

Kód FPGA je členěn do entit podle logických celků zprostředkovávajících jednotlivé funkce. Každý tento celek je samostatný blok s pevně definovaným rozhraním, jehož vnitřní logika je skryta z pohledu okolí. Příkladem entity ve fyzickém návrhu může být součástka registru na plošném spoji. Jelikož je návrh pro FPGA pouze teoretický zápis, může obsahovat libovolné množství entit. Záleží ovšem na fyzických vlastnostech čipu, zda-li je možné takový návrh do čipu vůbec naprogramovat.

4.2 Nejvyšší entita v hierarchii

Popis rozhraní entity je k dispozici v příloze B.1

Tato entita vytváří a propojuje instance všech ostatních entit v návrhu. Její rozhraní je přímo napojeno na piny pouzdra FPGA. V rámci návrhu také spravuje hodinový signál

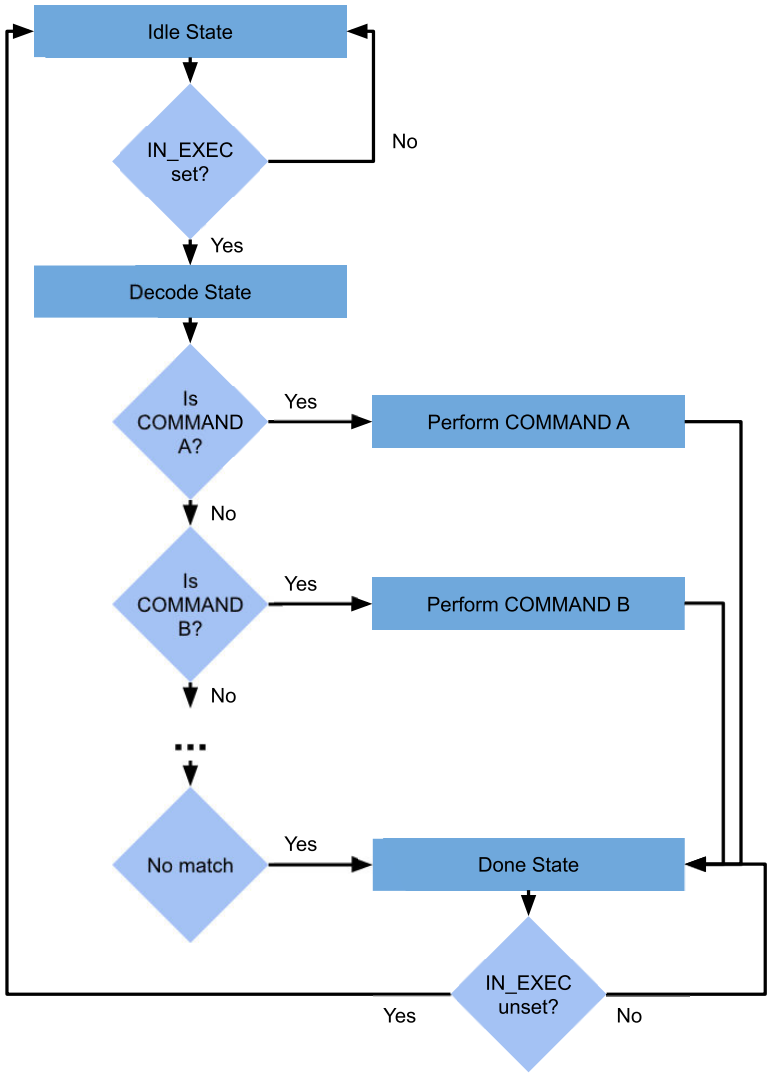
a zajišťuje asynchronní nastavení a synchronní vynulování signálu `reset` pro korektní fungování synchronní obnovení stavu logiky v celém návrhu.

V rámci hlavní entity je definována sada osmibitových registrů sloužících jako paměťový prostor pro příkazy a konfiguraci zapisovanou přes I2C. Pokud má konfigurační informace více bitů než 8, je několik registrů spojeno do jediného bitového vektoru o šířce 16 nebo 32 bitů. Informace zapsané do FPGA jsou tak neustále k dispozici ostatní logice, namísto nutnosti sekvenčního čtení z RAM. To velmi zrychluje celý návrh na úkor využití prostředků FPGA pro vytvoření registrů. Při zadání adresy pro čtení po I2C top modul přiřadí na vstup I2C modulu registry přímo z entit pro kontrolu ovladače krokových motorů.

Nejpodstatnější funkcí top entity je zpracování příkazů. Zpracování je realizováno stavovým automatem porovnávajícím `command` registr jednotky (jeden ze zmíněných osmibitových I2C registrů) s uloženými konstantami. Na základě výsledků těchto porovnání poté přechází mezi odpovídajícími stavy. Začátek porovnání je podmíněn signálem `IN_EXEC`, kterým mikrokontrolér oznamuje, že veškeré zápisy do registrů FPGA, včetně `command` registru, byly dokončeny a FPGA může provést příkaz. Následným porovnáním určí FPGA požadovaný příkaz a ten provede. Jednotka neopustí poslední stav `done` do momentu, než signál `IN_EXEC` klesne opět na log. 0. Poté přechází do výchozího stavu `idle`.

Kód stavového automatu následuje standardní konstrukci. Graf změn stavů je popsán v ilustraci 4.1. Jde o spojení synchronní a kombinační logiky—synchronní registr přenášející nový stav do aktuálního stavu vždy na tik hodin a kombinační logika volící následující stav na základě aktuálního stavu a ostatních signálů v entitě. Právě tato kombinační logika nastavuje vnitřní řídicí signály.

Ostatní entity v jednotce mění svůj stav na základě vstupních signálů. Pokud by byly I2C registry přímo napojeny na vstupy entit, změnil by se stav okamžitě po zápisu do registru, což by působilo problémy s vnitřní synchronizací jednotky při provádění příkazů. Jednotka proto obsahuje sadu mezi-registrů, které načtou data vždy až je prováděn konkrétní příkaz. To taky zvyšuje přehlednost kódu—registr uchovávající více bitových informací může být uložen do řady jedno-bitových signálů, čímž odpadne potřeba neustále indexovat vektor signálů.



Obrázek 4.1: Graf přechodů stavů konečného automatu nejvyšší entity.

4.3 Adresovatelný registr

Popis rozhraní entity je k dispozici v příloze B.2

Tato entita slouží zcela výhradně pro potřeby paměťových registrů v nejvyšší entitě FPGA. Implementace je prakticky identická se synchronním registrem, ovšem s přidáním komparátoru je možné registry adresovat pro zápis. Pokud adresa na bitovém vektoru `IN_ADDR` neodpovídá adrese vytvořené při syntéze registru, registr na zápis nereaguje, ani když je `IN_EN` signál nastaven. Všechny signály rozhraní kromě `OUT_Q` signálu jsou sdíleny mezi všemi registry.

4.4 Wishbone I2C master modul

Popis rozhraní entity je k dispozici v příloze B.3

Použitý model FPGA od společnosti Lattice obsahuje dva bloky pro komunikaci přes I2C přímo v hardware jako součást *Embedded Frame Block* (EFB). FPGA tak dokáže komunikovat po I2C bez nutnosti programovat samotnou logiku komunikace a se všemi výhodami této funkčnosti realizované v integrovaném obvodu, jako je vyšší rychlost a menší spotřeba. K zavedení této funkčnosti do FPGA se využívá systém *IPexpress* společnosti Lattice, který vygeneruje potřebný VHDL kód podle zadaných parametrů (frekvence zdrojových hodin, frekvence SCL signálu) a nakonfiguruje projekt aby využíval EFB na čipu.

Samotný I2C obvod na čipu je ovládán pomocí Wishbone sběrnice. Funkčnost tedy vyžaduje implementaci entity ovládající I2C komunikaci. To zaručuje velkou flexibilitu I2C obvodu a umožňuje programátorovi adaptovat komunikaci jak na oficiální specifikaci I2C, tak různé podmnožiny, či rozšíření, jako například *Two-Wire Interface* (TWI) implementaci používanou na čipech společnosti Atmel.

Wishbone sběrnice je navržena s ohledem na maximální flexibilitu. Specifikace podporuje řadu topologií a proto také obsahuje množství signálů nutných pro jejich realizaci [8]. V FPGA je využita pouze topologie propojující dvě zařízení typu master-slave a tím nutnost využití řady signálů odpadá. Wishbone protokol je v jádru založen na zápisu a čtení registrů slave zařízení a návrh se neobejde bez signálů, které to umožňují. Mezi tyto signály patří hodinový signál, signál resetování, datové signály, adresový signál, signál zápisu a potvrzovací signál. Signály hodin a resetování nejsou ovládány master zařízením, ale nejvyšší entitou jednotky. Proces zápisu nebo čtení slave zařízení spočívá v nastavení adresového signálu a čekání na potvrzovací signál. Když slave zařízení nastaví hodnotu potvrzovacího signálu na log. 1, na vstupním datovém signálu je platná hodnota z čtené adresy, nebo je na čtenou adresu zapsána hodnota na výstupním datovém signálu. Tyto dvě možnosti určuje signál čtení a zápisu.

4.4.1 I2C protokol

Pro komunikaci využívá FPGA I2C sběrnici popsanou v kapitole 3.3. I když základní protokol přenášení dat po I2C je definován specifikací, význam těchto dat je určen aplikací, která tento protokol využívá. Jednotka podporuje čtyři základní typy I2C komunikace—základní zápis a čtení hodnoty o velikosti 1 byte z adresy a sériový zápis a čtení hodnot o velikosti 1 byte. Při tomto zápisu provádí jednotka automatickou inkrementaci adresy. Zápis je jednoduchého formátu a spočívá v zapsání série bytů, při kterém FPGA interpretuje první byte jako cílovou adresu v zařízení, na kterou zapíše následující data. Proces čtení je kombinovaného formátu, tzn. během jedné komunikace se směr přenosu změní. Po prvním zapsaném bytu adresy v zařízení následuje čtení dat z této adresy.

Pokud na sběrnici neprobíhá komunikace, jsou oba vodiče na signálu log. 1. To umožňuje kterémukoliv master zařízení na sběrnici započít komunikaci. Začátek komunikace je definován specifickou změnou signálů na vodičích, tzv. *start condition*. Tento začátek je signalizován vynulováním SDA signálu před změnou SCL signálu. Následně je i SCL signál nastaven na log. 0 a slave zařízení začnou vzorkovat vstup z SDA vodiče na nástupnou hranu SCL signálu.

První byte je vždy odvysílán master zařízením při všech druzích komunikace. Obsahuje adresu slave zařízením v prvních 7 bitech, se kterým chce master komunikovat. Osmý bit je rozlišuje operaci čtení a zápisu. Pokud má tento bit hodnotu 0, master signalizuje zápis (W). Hodnota 1 znamená čtení (R). V řídicí jednotce první byte v komunikaci poslaný master zařízením vždy indikuje zápis.

Vždy následující bit po odvysílání jednoho byte je rezervován pro potvrzení. SDA i SCL vodiče jsou napojeny na V_{cc} přes pull-up odpory, master zařízení tak může SDA vodič uvolnit. Připojením vodiče na GND slave zařízení signalizuje úspěšné potvrzení. Pokud zůstane SDA vodič na vysoké úrovni napětí v dalším hodinovém taktu, master detekuje chybu komunikace a operaci přeruší.

Následující přenesený byte obsahuje adresu uvnitř zařízení, se kterou bude master chtít dále pracovat. Po bytu opět následuje potvrzení. Pro zjednodušení textu budou následující zmínky o potvrzení vynechány.

Pokud je záměrem master zařízení zapisovat, po adrese uvnitř zařízení následuje řetězec bytů, vždy potvrzovaný slave zařízením. Je na slave zařízení, jak se zachová v této situaci. FPGA je naprogramováno, aby vždy po přijatém bytu automaticky zvýšilo adresu zapisování o 1. Byty master zařízení zapisuje podle potřeby a celou transakci po potvrzení slave zařízením ukončí pomocí tzv. *stop condition* (viz dále).

Pokud je záměrem číst, musí master zařízení nejprve nastavit tzv. *repeated start condition*. Jde o identickou změnu signálů na vodičích jako při *start condition*, ovšem s chybějící *stop condition* po poslední *start condition*. Následuje opět 7 bitů adresy slave zařízení a bit čtení. Nyní je master zařízení připraveno z adresy uvnitř zařízení číst. Pokud slave zařízení komunikaci potvrdí, na hodiny se na SDA začnou objevovat bity z požadované adresy. Potvrzení následující po přečtení prvního byte však dává master zařízení a nikoli slave. Poté, dokud master potvrzuje přijatá data, slave zařízení posílá další. FPGA v tomto případě při čtení automaticky inkrementuje adresu uvnitř zařízení.

Když je komunikace dokončena, následuje tzv. *stop condition*. Jde o změnu na SCL a SDA signálech podobnou start condition, ovšem zrcadlově převrácenou—master uvolní hodinový signál a následně uvolní datovou linku.

4.4.2 Implementace v návrhu

Wishbone master, modul, který zprostředkovává řízení ostatních modulů na sběrnici, je realizován VHDL kódem založeném na vzorovém návrhu poskytnutém společností Lattice [4], sestávajícím ze stavového automatu. Ten ovládá slave zařízení (I2C obvod vestavěný v hardware) pomocí zápisů a čtení registrů zařízení. Data přicházející přes I2C jsou na hodinový signál vzorkována do bufferu I2C jádra, vyčítána stavovým automatem master zařízení a předávána dále do FPGA. Obrázek B.1 stručně ilustruje změny stavů stavového automatu.

4.5 Entita pro řízení krokových motorů

Popis rozhraní entity je k dispozici v příloze B.4

Tato nejdůležitější komponenta celého návrhu zajišťuje korektní nastavování signálů z FPGA do ovladačů krokových motorů. Modul sestává z bloků synchronní logiky propojené často signály a multiplexory.

Na nejvyšší úrovni je možné modul rozdělit do tří samostatných částí. Interně se v FPGA rozlišují dva druhy pohybu—pohyby využívající mechanismu rampování a pohyby, které tohoto mechanismu nevyužívají. Každý z těchto typů je řešen samostatnou částí entity. Oba tyto druhy poté mohou být řízené interním časovačem/počítadlem kroků, nebo mohou být aktivní po předem nespecifikovanou dobu. Poslední část vybírá na základě definovaných podmínek jednu z těchto dvou podsekcí, která má posílat signály do řadiče krokových motorů.

Pohyby nevyužívající rampování jsou jednodušší. Jedná se o pohyby spojené se sledováním (tracking) a korekcí (guide). Jejich hlavní charakteristikou je, že pokud jsou oba typy pohybu aktivní současně, rychlosti se sčítají namísto toho, aby se použila pouze jedna z nich. Mimo to mají pohyby nevyužívající rampování konstantní, maximální počet mikrokroků. To je pro použitou elektroniku řadiče motorů hodnota 128, které odpovídá hodnota "111"na tříbitovém vektoru. Tyto pohyby realizuje část entity ovládající krokové motory, interně nazvaná sledovací větev.

Pohyby, které rampování využívají jsou složitější, protože musí zahrnout celý mechanismus zvětšování a zmenšování rychlosti (viz 4.5.6) i mechanismus nastavování počtu mikrokroků odpovídajícího aktuální rychlosti. Jedná se o rychlosti volného pohybu (`move`) a přesunu na souřadnice (`goto`). Rampování vždy začíná od konkrétní, předem definované rychlosti. Pro jednotku je to centrovací rychlost (center). Pohyb touto rychlostí interně využívá rampování, ale okamžitě v prvním kroku dosáhne cílové rychlosti. Proto u nejnižší rychlosti ke změnám nedochází. Pohyby využívající rampování realizuje část entity, interně nazvaná pohybová větev.

4.5.1 Korekce (Guide)

Sekce řešící pulzní a automatickou korekci pozice obrazového pole je součástí sledovací větve. Prvním blokem sekce řídicí navádění je jednoduché počítadlo. To implementuje tzv. *pulse guide*, tedy počítačem řízenou pulzní korekci. Při zápisu se počítá se s definovaným taktem FPGA 50 MHz a zadaný čas `time` se hodnota `count` spočítá jako

```
count = time [ms] * 50000
```

která odpovídá době, po kterou čítač nastaví signál aktivace navádění. Jelikož je při nahrání potřeba určit také směr, který je nezávislý na standardním směru pohybu v registrech FPGA a předpokládá se, že převrácená hodnota času uložená v `count` nikdy nepřekročí hodnotu $2^{31} - 1$, je směr uložený jako MSB zapisované hodnoty, tedy

```
written_count = (dir == 1) ? count | (1 << 31) : count & ~(1 << 31)
```

a až poté je celá tato hodnota zapsána do registru FPGA po I2C. Příkaz PGD zapsaný do příkazového registru způsobí načtení směru z MSB bitu `pulse_guide` registru a dolních 31 bitů spojí nulovým bitem to 32-bitového vektoru a ten zapíše do registru čítače.

Výstupem čítače jsou signály směru a aktivace pulzní korekce. Navádění může být ale také automatické—tedy když připojená naváděcí kamera nastavuje jeden za 4 signálů směru na auto guider portu jednotky. Automatická korekce je aktivní po libovolně dlouhou dobu, dokud jsou nastaveny odpovídající signály pro směr a jednotka nemůže počítat s dobou, po jakou budou nastaveny. Tyto 4 signály jsou na nejvyšší úrovni zapojeny pomocí jednoduché logiky aby vytvořily jediný signál směru `IN_AGD_DIR` a jediný signál aktivity `IN_AGD_EN` pro modul řízení krokových motorů pro každou osu. Uvnitř jsou propojeny se signály z čítače pulzního navádění.

```
S_GUIDE_DIR <= IN_AGD_DIR when (IN_AGD_EN = '1') else S_PGD_DIR;
S_GUIDE_ACTIVE <= IN_AGD_EN or S_PGD_ACTIVE;
```

Pro další část modulu není podstatné, jestli je zdrojem signálu aktivního navádění a směru čítač, či externí zařízení ovládající korekci přes tzv. *autoguider* port jednotky.

Vstupem bloku je doba pulzní korekce, signál spuštění pulzního navádění a signály aktivity a směru automatického navádění. Výstupem je aktivita navádění a směr navádění.

4.5.2 Sledování (Tracking)

Sledovací sekce je jádrem sledovací větve.

Jméno sekce může být lehce zavádějící—z podstaty návrhu totiž tento modul neřeší pouze sledování, ale také korekci. Vstupem sekce jsou signály aktivity sledování a korekce, signál směru korekce a periody sledování a korekce, které definují rychlost střídání výstupního signálu. Výstupní signály jsou aktivita samotné sledovací větve, směr sledovací větve, a perioda čítače mikrokroku. Podle hodnot vstupních signálů aktivit se nastavují výstupní signály. Vstupní signály aktivit jsou dvě jedno-bitové hodnoty—signály `tracking_active` a `guide_active`—celkem tedy mohou nastat 4 situace:

- `tracking_active = '0' and guide_active = '0'`—Sledování i korekce jsou neaktivní. Výstupní signál aktivity sledovací větve je na nule. Výstupní perioda sledovací větve je na hodnotě rychlosti sledování. Tato hodnota je na bitovém vektoru periody i v momentu nulové aktivity z důvodu zajištění samospouštění. Hodnota signálu směru je nastavena na výchozí hodnotu sledování—tedy pohyb v záporném směru rektascenze.

- `tracking_active = '1' and guide_active = '0'`—Od předchozího stavu se hodnoty liší pouze v hodnotě výstupního signálu celé sledovací větve, který je na hodnotě logické 1. Tento stav je za dobu běhu jednotky nejčastější. Rychlost sledování je malá, nevyžaduje rampování a i při neočekávaném rozběhnutí v tomto stavu nehrozí riziko poškození.

- `tracking_active = '0' and guide_active = '1'`—Tento stav se vyznačuje tím, že nejvíce připomíná obyčejný pohyb, pouze konstantní rychlostí která nevyžaduje rampování ani změnu počtu mikrokroků. Hodnotě výstupního signálu celé sledovací větve je na logické 1. Signál směru však není konstantní, jako u ostatních stavů, ale určen hodnotami směru korekce, ať už automatická korekce, či MSB hodnoty čítače pulzní korekce. Výstupní vektor periody sledovací větve je nastaven na vstupní hodnotu periody korekce.

- `tracking_active = '1' and guide_active = '1'`—Tento stav se téměř neliší od stavu, kdy je pouze sledování aktivní. Jediným rozdílem je, že výstupní vektor periody sledovací větve neodpovídá periodě sledování, ale součtu period sledování a navádění, pokud je směr korekce nastaven na kladný směr rektascenze, nebo rozdílu period v opačném případě. Jinými slovy, pokud je aktivována korekce současně se sledováním, rychlost sledování se upravuje o rychlost korekce v závislosti na směru korekce.

4.5.3 Výstup sledovací větve

Výstup sledovací větve je okamžitě propagován do multiplexorů přepínající mezi sledovací a pohybovou větví. Jediný signál, který je připojen přes registr je signál aktivity sledovací větve. Je potřeba zajistit, aby byl signál nastaven zpět na nulu pouze v okamžiku dokročení celého kroku. Změny ostatních signálu, jako periody a směru jsou propagovány okamžitě.

4.5.4 Přesun na souřadnice (goto)

FPGA zprostředkovává nejnižší vrstvu abstrakce v celém procesu zpracování příkazů. Jako takové nemá jakoukoli představu o existenci nebeské sféry a souvisejících souřadnicích a úhlech. Z pohledu FPGA se pouze odehrává pohyb dvou nespecifikovaných motorů na

definovanou dobu, či o počet kroků. Proto i příkaz `goto` je v MCU přepočítán ze souřadnic na potřebný počet kroků, který těmto souřadnicím odpovídá, a teprve pak předán do FPGA.

Obdobně jako sledovací větev, i pohybová větev řeší dva typy pohybu—pohyb omezený čítačem a pohyb po dobu aktivního signálu. Na rozdíl od sledovací větve a čítače pro pulse guide, tento pohyb není omezený časem, nýbrž počtem celých kroků. Po aktivování čítače se nastaví jeho hodnota na hodnotu uloženou v registru a signál aktivity `goto` je nastaven na logickou 1. Oproti pulse guide čítači, který snižuje hodnotu na náběžnou hranu hodinového signálu, má `goto` čítač povolovací signál a snižuje svou hodnotu pouze na náběžnou hranu signálu dokončení celého kroku.

Vstupem čítače je vektor s hodnotou čítače a signál aktivace `goto` a signál celého kroku, který je interním signálem modulu. Na signál aktivace si čítač načte hodnotu ze vstupu do interního registru a začne odčítat celé kroky. Výstupem je signál aktivity navádění na souřadnice. Obdobně jako u sledovací větve, samotný mechanismus pohybu přesunu na souřadnice je realizovaný spolu s mechanismem volného pohybu v procesu implementujícím rampování.

4.5.5 Volný pohyb (Move)

Na rozdíl od navádění, `move` pohyb nepotřebuje interní čítač ani směr. Směr je společný se směrem využívaným pro `goto`, ale `move` jako pohyb má větší prioritu—pokud jednotka dostane příkaz provést `goto` v momentě, kdy je aktivní `move`, nenastane žádná změna, naopak příkaz `move` při aktivním `goto` automatický pohyb zruší.

K těmto specifikacím je `move` pouze jediný signál na vstupu entity. Není potřeba jej jakkoli propojovat s dalšími signály.

4.5.6 Rampování

Rampování je pravděpodobně nejsložitější a také nejdůležitější část celé entity. Zajišťuje lineární zrychlení a zpomalení při použití rychlostí zpravidla větších, než je naváděcí rychlost. Setrvačnost dalekohledu na montáži nedovoluje okamžité zrychlení na maximální rychlost, které by způsobilo buď přeskočení kroků motoru nebo v horším případě poškození mechaniky montáže. A jelikož má každá montáž jiné mechanické vlastnosti, musí být rampování plně nastavitelné.

S funkcí rampování je spojeno přepínání počtu mikrokroků při dosažení definovaných limitů rychlosti. Při nejnižších rychlostech (sledování, korekce) používá jednotka vždy maximální počet mikrokroků, tedy 128 pro použitý řadič, aby dosáhla co možná nejplynulejšího pohybu a co největšího úhlového rozlišení. Při maximálních rychlostech ale mikrokrokování nelze použít a krokový motor je potřeba ovládat celými kroky, jelikož mezní rychlost při použití mikrokrokování je podstatně menší než mezní rychlost když je motor řízen celými kroky. Proto je součástí konfigurace jednotky seznam limitů rychlosti, při nichž se počet mikrokroků vždy sníží na polovinu. Při snižování rychlosti jednotka opět při dosažení limitu zdvojnásobí počet mikrokroků.

Samotné rampování rychlosti se provádí změnou hodnoty v čítači délky kroku, která je na každý krok do čítače nahrána. Jelikož FPGA pracuje s dobou, za kterou je vykonán jeden krok, je třeba mluvit o periodě. Hodnota nahrávána do čítače musí být převrácenou hodnotou lineárně se měnící rychlosti. Dělení, i 1/x, je ale složitou operací na implementování v FPGA. Jelikož je použitý čip relativně malý, na implementaci dělícího algoritmu—ať už celé děličky, či *CORDIC* nebo jiného algoritmu—není v FPGA místo. Řešením je předem

propočítaná tabulka převrácených hodnot rychlostí. Během rampování pak stačí lineárně procházet tabulku a odpovídající hodnoty nahrávat do čítače.

Tabulka je v FPGA uložena v bloku integrované RAM. Při naplnění maximální kapacity zvolené FPGA podporuje dvě jedno-portové RAM, jednu pro jednu tabulku rampování pro každou osu, postavené na vestavěné blokové RAM o kapacitách 1024 položek, každá dlouhá 32 bitů.

Rampovací obvod ovládající čtení z RAM je postaven na jednom čítacím registru. Ten představuje ukazatel do tabulky uložené v RAM. Rampování je dosaženo lineárním zvětšováním a zmenšováním hodnoty registru. K ovládání registru slouží tři signály—zvětšení, zmenšení a vynulování—které jsou nastavovány hlavním procesem logiky rampování. Vždy na náběžnou hranu hodin provede registr odpovídající operaci na základě hodnot těchto signálů. Výstup registru je přímo naveden na vstup adresy paměti RAM a změna hodnoty se se zpožděním jednoho taktu projeví na výstupu paměti, který je přímo napojen zpět na modul ovládání krokových motorů. Samotné lineární zvětšování a zmenšování neprobíhá na takt, ale je podmíněné aktivačním signálem pro možné nastavování rychlosti rampy. Ten je realizován děličkou signálu vytvořenou akumulátorem a multiplexorem. Na náběžnou hranu výstupního signálu je provedeno zvýšení či snížení hodnoty čítače.

Obsah paměti RAM obsahuje periodu odpovídající požadované rychlosti i počet mikrokroků s touto rychlostí spojený. K jednoduchému generování této tabulky byl napsán krátký skript v jazyce Python. Detaily o obsahu paměti i skriptu jsou popsány v kapitole 4.5.7.

Druhý čítač v rampovací logice počítá počet kroků, po který se rampovalo vzhůru. Tento čítač je využívám pouze při najíždění na souřadnice. Pokud se zbývající počet kroků do cíle rovná hodnotě čítače, je potřeba začít rampovat zpět k nejnižší rychlosti. Tato potřeba může nastat ve dvou případech:

- Jednotka dosáhne nejvyšší rychlosti a pokračuje v pohybu. Určitý počet kroků před cílovými souřadnicemi je ale potřeba opětovně začít zpomalovat. Tento počet kroků se rovná počtu kroků potřebnému pro zrychlení na maximální rychlost.
- Jednotka nestihne dosáhnout nejvyšší rychlosti před dosažením poloviny celkové vzdálenosti. Pak musí začít opět zpomalovat, aby nepřejela koncový bod.

Návrh mechanismu dovoluje řešit oba tyto případy současně. Při zrychlování, pokud není dosaženo maximální rychlosti, se čítač počtu kroků rampujících vzhůru zvětšuje. Pokud v kterémkoli momentě hodnota tohoto čítače odpovídá zbývajícímu počtu kroků čítače pro najíždění na souřadnice, jednotka začne okamžitě zpomalovat, dokud nedosáhne nominální rychlosti.

Tuto problematiku není nutné řešit při volném pohybu, obzvláště proto, že ji řešit ani není možné. Jednotka nezná předem dobu, po kterou se bude pří volném pohybu pohybovat. Volný pohyb při aktivaci začne zrychlovat až do maximální rychlosti. Při klesnutí signálu `move_active` na log. 0 začne plynule zpomalovat.

Čítač ukazatele do tabulky a čítač počtu kroků musí být dvě samostatné logiky, jelikož je každý upravovaný na náběžnou hranu jiného signálu. Ukazatel do tabulky má vlastní děličku signálu. Počítadlo kroku je aktivní na náběžnou hranu celého kroku.

4.5.7 Obsah RAM a jeho generování

Architektura jednotky využívá RAM pro uložení tabulky period. V průběhu rampování je potřeba rychlost zvyšovat nebo snižovat lineárně. Pro získání periody, která je použí-

vána mechanismem provádějícím kroky z rychlosti je ale zapotřebí poměrně náročné logiky. Propočítání předem této tabulky a následné lineární procházení touto tabulkou je značně úspornější. Rychlost se při použití tabulky sice nemění lineárně, ale po diskrétních krocích, to ale není pro návrh omezující. Stejná tabulka je využívána také k nastavení počtu mikrokroků odpovídajícímu aktuální rychlosti.

Tabulka rychlosti obsahuje 1024 hodnot. Lineární interpolací se spočítá 1024 rychlostí mezi zadanou minimální a maximální rychlostí v krocích za sekundu. Z těchto rychlostí se vypočítá jejich převrácená hodnota násobená taktem hodin FPGA—výsledkem je počet taktů, které je potřeba čekat mezi kroky pro dosažení odpovídající rychlosti.

Jelikož je počet mikrokroků vázaný na periodu a veškeré možné periody jsou známy, není třeba porovnávat periodu na uložené konstanty v registrech, ale je možné ji propočítat předem pro všechny možnosti. To ušetří logiku potřebnou pro syntézu řady komparátorů, a zrychlí celý systém, protože není třeba čekat jeden (či více) hodinových taktů na dekódování mikrokroků.

Počet mikrokroků je zakódován jako horní tři bity u každé položky. Perioda každé rychlosti pak nesmí přesáhnout rozsah 29 bitů. Horní tři bity jsou nahrazeny počtem mikrokroků jako $log_2(n)$.

4.5.8 Výstup pohybové větve

Výstup signálu aktivity pohybové větve je řešen stejně, jako výstup signálu aktivity větve sledovací—výstup je nastaven na log. 1 kdykoli je vstupní signál aktivní, ale na log. 0 se signál shodí pouze při nástupné hraně signálu celého kroku.

Ostatní signály, jako výstupní perioda, směr a mikrokroky, jsou vzorkovány přes registry vždy na nástupní hranu signálu celého kroku. Tyto hodnoty nelze ze specifikace měnit mezi celými kroky pro korektní fungování systému.

4.5.9 Čítač mikrokroků, detekce a čítač celých kroků

Poslední blok entity řídicí krokové motory je tvořen z několika částí. První částí je samotný čítač mikrokroku. Jedná se o jednoduchý registr, který v okamžiku, kdy je jeden z interních signálů entity—aktivita sledovací, či pohybové větve—aktivní, snižuje hodnotu čítače na krok hodin. Pokaždé, kdy je hodnota rovna nule je signál mikrokroku nastaven na log. 1, ve všech ostatních případech je jeho hodnota nula. V tomtéž kroku také dojde k nahrání nové periody do čítače. Hodnota je logicky posunována vpravo podle aktuální hodnoty interního sígnálu počtu mikrokroků. Počtu mikrokroků n odpovídá hodnota signálu $log_2(n)$. Posun o $log_2(n)$ bitů vpravo pak odpovídá dělení hodnoty číslem n.

Periody, ale také ostatní signály směru a mikrokroků, jsou na základě signálu aktivity pohybové větve vybírány buď z pohybové větve, pokud je tato aktivní, nebo ze sledovací větve. Důsledkem tohoto chování je priorita pohybové větve. Pokud je aktivní, stav sledovací větve začne je ignorován, ale také se nemění—při deaktivaci pohybové větve se jednotka vrátí do stavu určeného sledovací větví.

Velmi důležitou částí, podle které se synchronizuje zbytek entity, je čítač celého kroku. Ten je tvořen jednoduchým čítačem, který je vždy na mikrokrok inkrementován. Jelikož je počet mikrokroků vždy n-tá mocnina, celý krok se projevuje změnou n-tého bitu čítače (číslováno od nuly, počínaje LSB). Tento bit je ale nastaven ne na jeden takt hodin, ale na polovinu celého kroku a tudíž je pro jeho použití nutné detekovat náběžnou hranu.

Signál nemůže být použit v tzv. *sensitivity seznamu* procesu. To implikuje syntetizujícímu programu zavedení nové časové domény. Pro detekci hrany je potřeba použít obvodu detekujícího náběžnou hranu.

```
examined_signal_d <= examined_signal when rising_edge(clock);
examined_signal_re <= not examined_signal_d and examined_signal;
```

Signál s názvem `examined_signal` je signál, jehož náběžnou hranu je třeba detekovat. Ten je na hodiny vzorkován registrem, jehož výstupem je `examined_signal_d`. Stejného mechanismu by bylo možné docílit synchronním procesem, na hodiny přiřazujícím vzorkovaný signál do výstupního signálu. Následuje kombinační logika, porovnávající původní signál s negací vzorkovaného signálu. Výsledkem je signál aktivní pouze pokud je `examined_signal` na log. 1 a `examined_signal_d` na log. 0.

Kapitola 5

Návrh firmware pro MCU

Mikrokontrolér představuje jádro jednotky. Podílí na veškerých výpočetních úkonech, zajišťuje komunikaci s připojeným PC a zpracování příkazů, správu a ukládání konfiguračních i stavových dat, komunikaci s RTC modulem a nastavování digitálně-analogového převodníku. Funkce vestavěné v hardware a podporované v knihovnách Advanced Software Framework 4 (dále ASF) dovolují snadno využít i složitější funkčnost poskytovanou zabudovanými prostředky mikrokontroléru, jako je například I2C pro komunikaci s FPGA, USART, Ethernet a USB pro komunikaci s připojeným PC, ale i matematické knihovny optimalizované pro AVR, či časová přerušení využívaná zcela uvnitř mikrokontroléru [6].

Technické specifikace MCU jsou ivedeny v oficiální dokumentaci [1].

5.1 Stav jednotky a hlavičkový soubor `common.h`

Jednotka si musí uchovávat množství informací potřebných pro korektní fungování, propočty a předávání informací připojenému PC, či ovládací ručce. Tyto informace jsou za běhu uloženy v datové části programu. Aby bylo jednodušší data ukládat a načítat, má jednotka definované dvě velké datové struktury, dále složené z menších datových typů. Informace jsou do struktur rozděleny podle toho, zda-li má jednotka možnost informace odvodit z jiných, známých informací, či jestli musí být nastavena uživatelem. Odvoditelná data není třeba uchovávat a jsou uložena ve struktuře `state_data_t`. Data nastavována uživatelem se ukládají do struktury `configuration_data_t`. Ta je při nečinnosti jednotky uložena do paměti FLASH vestavěné do mikrokontroléru a z této paměti jsou data po zapnutí nahrávána.

```
typedef struct {
    motor_config_t ra_motor;
    motor_config_t dec_motor;
    encoder_config_t ra_encoder;
    encoder_config_t dec_encoder;
    user_speed_table_t speed_table;
    user_period_table_t period_table;
    speed_ids_table_t speed_ids_table;
    microstep_table_t mstep_table;
    current_config_t current_config;
    double_vec2_t location;
    double_t sin_lat;
```

```
    double_t cos_lat;
    double_vec2_t park_coords;
    uint8_t alignment;
    uint8_t tracking_state;
    uint8_t pier_side;
    uint8_t hemisphere;
} configuration_data_t;

typedef struct {
    bitset16_t unit_state;
    bitset16_t update_status;
    uint8_t ra_mov_state;
    uint8_t dec_mov_state;
    uint8_t mov_speed_id;
    double_t sync_time;
    double_vec2_t sync_coords;
    tracking_period_table_t tracking_period_table;
    double_t tracking_speed_sidereal;
} state_data_t;
```

Dvě struktury typu `motor_config_t` udržují informaci o nastavení motorů, jako jsou kroky na otáčku a bit inverze pohybu. Ten umožňuje upravovat nastavení směru každého motoru podle konstrukce montáže. Dvě struktury typu `encoder_config_t` uchovávají informaci o konfiguraci enkodérů, jako je typ/přítomnost enkodéru pro danou osu a počet pulsů na otáčku. Struktura typu `user_speed_table_t` obsahuje pole všech uživatelem konfigurovaných rychlostí a struktura typu `user_period_table_t` obsahuje pole převrácených hodnot těchto rychlostí. Ve struktuře typu `microstep_table_t` jsou uloženy limity rychlostí používané při výpočtu tabulky rampovací tabulky period zapisované do FPGA. Poslední struktura typu `current_config_t` obsahuje tři hodnoty proudů pro konfiguraci DAC. Tyto hodnoty jsou využívány při změně proudu do motorů jednotlivých os. Struktura typu `double_vec2_t` představuje dvojici čísel v plovoucí řádové čárce. Položka konfigurace `location` obsahuje zeměpisné souřadnice montáže a položky `sin_lat` a `cos_lat` sinus a cosinus zeměpisné šířky pro urychlení následných výpočtů. Položka `park_coords` obsahuje souřadnice v horizontálním systému pro najetí při příkazu parkování 3.5.7. Položka `alignment` udržuje informaci o typu montáže—hodnoty mohou být azimutální montáž, rovníková vidlicová, či rovníková německá 2.4. Položka `tracking_state` udržuje informaci o rychlosti/zapnutí sledování 3.5.2. Položka `pier_side` určuje stranu od pilíře u německých montáží 3.5.6. Poslední položka `hemisphere` v případě nastavení jižní polokoule invertuje směry pohybu motorů.

Ve struktuře stavových dat je udržována bitová množina `unit_state` obsahující informace o stavu jednotky, jako je například přesunu na souřadnice a položka `update_status` obsahuje informace o změně stavových informací, které jsou získatelné ostatními příkazy protokolu (`get location`, apod.). Připojené PC tyto informace pravidelně požaduje v příkazu `get state` pro neustálé zobrazovaní aktuálních dat uživateli—tyto informace mohly být změněny jiným PC, či jednotkou ručního ovládání a jednotka musí umět tuto informaci indikovat. Položky `ra_mov_state` a `dec_mov_state` obsahují aktuální stav pohybu pro obě osy. Položka `sync_time` obsahuje juliánské datum poslední synchronizace a položka `sync_coords` obsahuje rovníkové souřadnice přijaté z PC při synchronizaci. Struktura typu `tracking_period_table_t` obsahuje všechny předem vypočítané sledovací periody. Jed-

notka si tuto informaci může pokaždé po zapnutí vypočítat z konfigurace motorů. Poslední položka `tracking_speed_sidereal` slouží výhradně pro urychlení interních výpočtů.

Všechny datové struktury jsou definovány v jediném hlavičkovém souboru `common.h` umístěném v kořeni adresáře `components` namísto vlastní složky uvnitř tohoto adresáře. Struktury jsou přidané do každého zdrojového souboru jednotky.

5.2 Hlavní smyčka a funkce `main()`

Program obsahuje ve funkci `main()` nekonečnou smyčku, což je typická konstrukce používaná v oblasti vestavěných systémů. Hlavní funkce je složena ze dvou částí—inicializace, kde se provedou všechny potřebné kroky pro nastavení prostředků mikrokontroléru, načtení konfiguračních dat a příprava na přijetí příkazu, a samotná nekonečná smyčka, ve které jednotka čeká na příchozí příkaz a poté provede jeho zpracování.

Jednotlivé úkoly, které jsou v hlavní funkci prováděny, dobře ilustrují stavy programu prováděného jednotkou, od inicializace až po odeslání odpovědi na přijatý příkaz.

5.2.1 Inicializace

Před veškerými operacemi prováděnými v programu je potřeba inicializovat hardware jednotky. Inicializace hardware se provádí voláním funkcí, které jsou součástí ASF pro každou hardware komponentu. Prvním krokem je inicializace GPIO pinů na odpovídající vstupy/-výstupy, přerušení po GPIO a pull-up či pull-down rezistory u vstupů. Inicializace GPIO je pouze volání zápisů do registrů a není předpokládáno, že by mohla selhat. Prvním kritickým krokem inicializace je nastavení hodin mikrokontroléru. V případě selhání rozsvítí jednotka červenou LED a vstoupí do prázdné nekonečné smyčky. Takovéto chování je běžné i v dalších částech programu—pokud selže klíčový hardware, další funkčnost nemá smysl. Dále jsou nakonfigurovány všechny hardware komponenty jednotky—USART a TWI. V rámci těchto operací jsou nakonfigurovány odpovídající vektory přerušení a následně jsou všechna přerušení aktivována. Posledním krokem inicializační rutiny je resetování FPGA nastavením signálu pro resetování na log. 1 a zpět.

Nasleduje software inicializace. Volání funkce `memset()` inicializuje veškerou paměť obsahující stavová a konfigurační data na hodnotu 0. Poté se jednotka pokusí přečíst konfigurační data z uživatelské FLASH paměti v MCU. Pokud se jednotce podaří načíst data, pokusí se ověřit jejich konzistenci. Pokud data nejsou načtena, jednotka zavolá funkci nastavující konfigurační data na výchozí hodnoty. Tento mechanismus je přítomný ve vývojové verzi a záleží na zpětné vazbě uživatelů, zda-li je podobné chování žádoucí.

S konfiguračními daty v paměti program přejde na inicializaci jednotky a FPGA. Inicializace jednotky spočívá v propočtu stavových dat z dat konfiguračních. To je hlavně sledovací hvězdná rychlost a sledovací periody pro zapsání do FPGA. Tyto hodnoty se přímo odvíjí od hodnot nastavení kroků na otáčku motoru rektascenze. Konfigurace FPGA sestává z nahrání všech potřebných hodnot z konfiguračních a stavových dat do registrů FPGA přes I2C.

5.2.2 Čekání na FPGA

Začátek hlavní smyčky obsahuje test signálu na GPIO vstupu mikrokontroléru, kterým FPGA signalizuje, že je připraveno zpracovat příkaz. Podmínka testuje hodnotu na GPIO

vstupu. Pokud je hodnota na log. 0, jednotka se vrací na začátek smyčky. Čekání (které zpravidla indikuje poruchu FPGA) je indikováno stálým svícením zelené LED.

5.2.3 Načtení příchozího příkazu

Komunikace s připojeným PC je realizována pomocí přerušení. Při přijetí příkazu přes kterýkoli způsob komunikace nastaví přerušení příznak přijatého příkazu. Jednotka tyto indikátory sekvenčně testuje. První indikátor, který je nastaven na `true`, vyvolá překopírování příchozího bufferu do globální proměnné `input_buffer`. Následně jednotka nastaví potřebné pomocné proměnné, jako je proměnná `CB_index` uchovávající délku příchozího bufferu a proměnnou `command_origin`, která slouží k rozlišení původu příkazu.

Volání funkce pro přečtení příkazu z přijatého bufferu C.3.2 s parametry `input_buffer` a `CB_index` vrátí hodnotu rozpoznaného příkazu, případně hodnotu poškozeného či neznámého příkazu. Jakmile je znám příkaz, jednotka vynuluje globální proměnnou uchovávající délku odpovědi `response_length` a také paměť bufferu `response_buffer`, které jsou používané pro sestavení a odeslání odpovědi.

5.2.4 Zpracování příkazu

Pokud je výstup funkce pro načtení příkazu z bufferu hodnota chyby (neznámý, či poškozený příkaz), jednotka pomocí odpovídajících funkcí C.3.5 sestaví jednoduchou odpověď.

Pokud je přijatý příkaz platný, sestavení odpovědi je součástí funkce vykonávající odpovídající příkaz. Rozskok podle hodnoty rozpoznaného příkazu je proveden ve dvou `switch` blocích. Vnější blok zachytí konfigurační příkazy. Vnitřní blok je vnořen pod `default` možnost vyššího bloku a provede se pouze pokud je jednotka správně nakonfigurována.

Jednotlivé případy příkazu `switch` poté volají funkce odpovídající jednotlivým příkazům. Každý příkaz komunikačního protokolu, ať už konfigurační či obecný, má přiřazenu právě jednu funkci. Ta provádí veškeré úkony spojené s tímto příkazem, včetně sestavení odpovědi. Detailní popisy jednotlivých funkcí jsou rozepsány v dokumentaci zdrojového kódu C.3.3.

5.2.5 Odeslání odpovědi

Posledním funkčním blokem jednotky je odeslání odpovědi. Ta je sestavena v globální `response_buffer` proměnné některou z funkcí volaných při provádění příkazu. Na základě `command_origin` proměnné je odpověď odeslána odpovídající funkcí některé ze tří knihoven pro komunikaci s PC. Pokud je zdrojem příkazu sériová linka, větev odesílající odpověď ověří, zda-li nebyl přijat další příkaz do odeslání odpovědi. Pokud ano, jednotka odešle dodatečnou `!OVR^` odpověď. Poté jednotka vynuluje příznak přijatého příkazu pro zdroj, případně využije ASF procedury pro vyčištění příchozího bufferu, který právě zpracovala. Teprve to umožní přijetí nového příkazu. Samostatné příznaky přijetí příkazu pro každý způsob komunikace dovolují zpracování příkazů přicházejících z různých zdrojů ve stejný čas. Rizikem tohoto přístupu je vyhladovění některého ze zdrojů příkazů. Příznaky jsou zpracovávány s pevně danou prioritou určenou pořadím v `if` bloku a pokud je některý z výše umístěných způsobů komunikace příliš aktivní, nemusí dojít ke zpracování ostatních příkazů. K této situaci by ovšem za normálních podmínek nemělo dojít.

5.3 Funkce zpracovávající příkazy

Zpracování každého příkazu, který může připojené zařízení poslat do jednotky, je implementováno svou vlastní funkcí, která provádí všechny operace spojené s vykonáním daného příkazu. To zahrnuje vyčtení parametrů z přijatého bufferu, zpracování, uložení či vyčtení požadovaných dat a následně sestavení odpovědi. Funkce odpovídá za správné provedení operace a zajišťuje navrácení chyby, pokud nějaká nastane.

Všechny tyto funkce nemají návratovou hodnotu. Reakce na chybu spočívá v nastavení příslušných hodnot v návratovém bufferu `response_buffer`, jehož počet znaků, definovaný proměnnou `response_length` je vždy na konci hlavní smyčky odeslán. Reakce na chybu je tedy realizována naplněním bufferu odpovídajícími hodnotami a předčasným opuštěním funkce. Integrita vnitřních dat jednotky je zaručena návrhem funkcí. Veškeré načtení a změny jsou prováděny nad lokálními proměnnými a do paměťové struktury se zapisují až po skončení všech operací, které mohou selhat. Odpověď je sestavena vždy na začátku funkce podle formátu v tabulce Definice obecných příkazů komunikačního protokolu A.1. Jakýkoliv neúspěch v provádění funkce přepíše první znak bufferu na symbol `!` a pokud funkce navracela data, tedy odpověď byla delší, než 5 znaků, funkce přepíše pátý znak bufferu na symbol ukončující řetězec `^` a nastaví `response_length` na 5. Tyto operace jsou umístěny za příkazem `return` ve funkci a jsou uvedeny návěštím `FAILURE`, na které program skočí v případě chyby.

Všechny chyby vrací protokolem chybu. Z chyb, jako je chyba ve formátu přijatého příkazu nebo parametrů příkazu, či příkaz neproveditelný v současný moment, se lze zotavit. Chyby hardware, jako například selhání komunikace s FPGA, či RTC modulem, jsou považovány za kritické. V případě takovéto chyby nastaví jednotka globální příznak `critical_failure`, který zajistí zamrznutí programu jednotky na konci hlavní smyčky. Toto zamrznutí je doprovázeno blikáním červené LED pro indikaci uživateli. V případě kritické chyby je nutná oprava jednotky.

5.4 Komunikace s FPGA

Komunikace s FPGA čipem na desce probíhá přes dvojici rozhraní. Prvním z nich je přímé propojení GPIO pinů MCU a FPGA, což je nejjednodušší a nejrychlejší způsob přenášení bitových informací pomocí logických signálů. Druhým způsobem je komunikace přes I2C, která využívá standardního protokolu definovaného I2C standardem. V FPGA tuto funkcionalitu zajišťuje vestavěná logika, která je součástí čipu a je popsána v kapitole FPGA 4.4.1.

V mikrokontroléru zajišťuje obsluhu hardware pro přenos dat přes I2C knihovna z ASF. Pomocí funkcí přijímající strukturu definovanou v načtené knihovně je možné posílat a číst řetězce dat z ostatních zařízení na připojené I2C sběrnici C.6 C.8.

Kapitola 6

Závěr

Cílem práce bylo vytvořit samostatnou jednotku pro použití ve vědeckém astronomickém pozorování a astrofotografii, která je schopna komunikovat s připojeným PC a využívá hardware prostředků na plošném spoji (MCU + FPGA). Jednotka dokáže sledovat objekty na obloze různými rychlostmi, korigovat odchylku těchto objektů v obraze i během sledování, provádět volný pohyb uživatelem definovanými rychlostmi a přejíždět na zadané souřadnice. Rychlejší pohyby jsou postupně zrychlovány z minimální do požadované rychlosti a zpět. Jako součástí práce vznikl také vlastní protokol pro komunikaci s připojeným PC a konfiguraci jednotky.

I když jednotka nabízí minimální sadu funkcí potřebných pro pozorování, je zapotřebí jednotku rozšířit o další funkčnost, vedoucí na vyšší komfort uživatele při obsluze jednotky. Podstatné je obzvláště dříve zmíněné ukládání konfiguračních dat do FLASH paměti v MCU, překlápění dalekohledu na německých montážích a parkování na uložené horizontální souřadnice. Dalšími kroky je rozšíření možností komunikace jednotky o USB a Ethernet rozhraní. Implementace Ethernet rozhraní pak otevírá další možnosti, jako vzdálené ovládání jednotky z místní sítě, či celého internetu a také snadné rozšíření ovládacích zařízení—např. mobilní aplikace využívající standardního rozhraní, komunikující s jednotkou.

Pro mne byla práce jak zajímavým úvodem do základů robotiky, tak uvedením do profesního života. Cílem nebyla cvičení bez praktického využití, ale reálný, fungující produkt. Řešení reálných problémů také vyžaduje komunikaci s koncovým uživatelem a tvůrce práce si již nevystačí s vlastními nápady, což je cenná zkušenost.

V práci bych chtěl pokračovat dále. Přes to, kolik toho jednotka umí, je spousta funkcí a detailů, které v práci realizovány nejsou. S dostatkem času věřím, že může vzniknout reálný produkt žádaný mezi uživateli.

Literatura

[1] Atmel Corporation, San Jose, California, United States: *Dokumentace 32UC3A rodiny MCU* . [Online; navštíveno 05.05.2019].
URL http://ww1.microchip.com/downloads/en/DeviceDoc/doc32058.pdf

[2] Couper, H.: *Dějiny astronomie* . Praha: Knižní klub, první vydání, 2009, ISBN 978-80-242-2367-4.

[3] Lattice Semiconductor, Portland, Oregon, United States: *Dokumentace MachXO2 rodiny FPGA* . [Online; navštíveno 30.07.2018].
URL https://www.latticesemi.com/~/media/LatticeSemi/Documents/DataSheets/MachXO23/MachXO2FamilyDataSheet.pdf

[4] Lattice Semiconductor, Portland, Oregon, United States: *Vzorový návrh Wishbone master modulu pro I2C* . [Online; navštíveno 12.05.2019].
URL https://www.latticesemi.com/-/media/LatticeSemi/Documents/ReferenceDesigns/EI/I2CMasterwithWISHBONEBusInterface-Documentation.ashx?document_id=32340

[5] Meeus, J.: *Astronomical algorithms* . Richmond, Va.: Willmann-Bell, druhé vydání, 1998, ISBN 0943396611.

[6] Microchip Technology Inc., Chandler, Arizona, United States: *Dokumentace Advanced Software Framework 4 (ASF)* . [Online; navštíveno 09.05.2019].
URL https://asf.microchip.com/docs/latest/index.html

[7] NXP Semiconductors, Eindhoven, Netherlands: *Definice sběrnice Wishbone* . [Online; navštíveno 05.05.2019].
URL https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[8] Silicore Corporation: *Dokumentace sběrnice Wishbone* . [Online; navštíveno 05.05.2019].
URL http://cdn.opencores.org/downloads/wbspec_b4.pdf

[9] Vanýsek, V.: *Základy astronomie a astrofyziky* . Praha: Academia, první vydání, 1980.

Příloha A

Příkazy komunikačního protokolu

Sloupec *Příkaz* obsahuje obecné jméno příkazu. Sloupec *Kód* obsahuje kód příkazu. Tímto kódem je identifikován příkaz ve zprávách protokolu. Sloupec *Popis* obsahuje textový popis příkazu. Nejpodstatnější je u prvních dvou položek, kde název příkazu není dostatečně popisný. Sloupec *Formát požadavku* popisuje formát příkazu v protokolu. V popisu je použito několik speciálních znaků, konkrétně *s*, *d* a *h*. Znak *s* reprezentuje znaménko a v protokolu může být na jeho místě buď znak *+* nebo znak -. Znak *d* představuje číslici v dekadické soustavě, tedy takový znak splňující regulární výraz `[0-9]`. Znak *h* představuje číslici v hexadecimální soustavě, tedy takový znak splňující regulární výraz `[0-9a-fA-F]`. Hranaté závorky v zápisu oddělují jednotlivé parametry příkazu pouze visuálně a protokolem se nepřenáší. Číslo v zápisu značí určitý počet znaků typu, který následuje za číslem. Tečka nemá speciální význam a protokol očekává znak tečky. Sloupec *Parametry* obsahuje jméno parametrů příkazu. Pořadí parametrů uvedených ve sloupci odpovídá pořadí skupin hranatých závorek ve předchozím sloupci. Sloupec *Formát odpovědi* obsahuje formát odpovědi. Formát je identický formátu sloupce *Formát požadavku*, s přídavkem dalších znaků. Znak / ve skupině hranatých závorek značí alternativu jedné ze dvou možností. V tomto případě se jedná o znaky *!* a tečku. Tečka na začátku odpovědi znamená pozitivní odpověď, tedy, že příkaz byl přijat a proveden úspěšně. Pokud se kdekoliv během provádění příkazu vyskytne chyba, jednotka nahradí znak tečky znakem *!* značícím negativní odpověď.

Tabulka A.1: Definice obecných příkazů komunikačního protokolu

Příkaz	Kód	Popis	Formát požadavku	Parametry	Formát odpovědi
?		Neznámý příkaz	Jakýkoliv nerozeznaný		!UCMˆ
overrun		Nový požadavek byl doručen před zpracováním předchozího	Kterýkoliv známý příkaz		!OVRˆ
no operation	NOP	Testování spojení přes COM a ETH	$NOPˆ	-	.NOPˆ
get version	VER	Vrátí verzi firmware ve formátu major.minor.build.revision	$VERˆ	-	.VER[8h]ˆ
select mov speed	SSP	Nastaví rychlost volného pohybu	$SSP[2h]ˆ	speed	[.|!]SSPˆ
start move	MOV	Začne volný pohyb v daném směru	$MOV[2h]ˆ	direction	[.|!]MOVˆ
stop move	STP	Vypne volný pohyb v daném směru	$STP[2h]ˆ	direction	[.|!]STPˆ
goto ra/dec	GTO	Přesun na souřadnice RA, Dec [rad]	$GTO[1d.6d][s1d.6d]ˆ	right_ascension, declination	[.|!]GTOˆ
stop goto	SGT	Deaktivuje probíhající přesun na souřadnice	$SGTˆ	-	[.|!]SGTˆ
synchronize ra/dec	SYN	Synchronizuje jednotku na RA, Dec [rad]	$SYN[1d.6d][s1d.6d]ˆ	right_ascension, declination	[.|!]SYNˆ
flip pier side	FPS	Překlopí montáž ve vidlicovém módu	$FPSˆ	-	[.|!]FPSˆ
set jd	SJD	Nastaví jednotce juliánské datum	$SJD[7d.8d]ˆ	julian_date	[.|!]SJDˆ

Tabulka A.2: Definice obecných příkazů komunikačního protokolu, pokračování

Příkaz	Kód	Popis	Formát požadavku	Parametry	Formát odpovědi
set location	SLC	Nastaví jednotce zeměpisné souřadnice	$SLC[1d.6d][s1d.6d]^	latitude, longitude	[.|!]SLC^
get state	GST	Přečte stav jednotky	$GST^	-	.GST[16h]^
get RA, Dec	GRD	Přečte nebeské souřadnice dalekohledu	$GRD^	-	[.|!]GRD[1d.6d][s1d.6d]^
get location	GLC	přečte zeměpisné souřadnice jednotky	$GLC^	-	.GLC[1d.6d][s1d.6d]^
get jd	GJD	Přečte juliánské datum jednotky	$GJD^	-	[.|!]JD[7d.8d]^
set hemisphere	SHE	Nastaví jednotce zemskou polokouli	$SHE[2h]^	hemisphere	[.|!]SHE^
set pier side	SPS	Nastaví jednotce stranu od pilíře	$SPS[2h]^	pier_side	[.|!]SPS^
pulse guide	PGD	Pohne dalekohledem daným směrem po daný čas	$PGD[4h][4h]^	ra_time, dec_time [ms]	[.|!]PGD^
set park	SPK	Nastaví aktuální souřadnice jako parkovací souřadnice	$SPK^	-	[.|!]SPK^
park	PRK	Přesune jednotku na parkovací souřadnice	$PRK^	-	[.|!]PRK^
unpark	UPK	Obnoví souřadnice ze zavolání park	$UPK^	-	[.|!]UPK^
abort	AAA	Okamžitě zastaví všechny pohybové operace	$AAA^	-	[.|!]AAA^

Tabulka A.3: Definice konfiguračních příkazů protokolu

Příkaz	Kód	Popis	Formát požadavku	Parametry	Formát odpovědi
configure motor entry	CMR	Nastaví konfiguraci položky motoru	$CMR[2h][8h][2h]^	motor_id, revolutios, dir	[.|!]CMR^
configure encoder entry	CER	Nastaví konfiguraci položky enkodéru	$CER[2h][8h][2h]^	encoder_id, pulses_per_rev, present_or_type	[.|!]CER^
configure speed entry	CSD	Nastaví konfiguraci položky rychlosti	$CSD[2h][8h]^	speed_id, steps_ps	[.|!]CSD^
configure mstep entry	CMS	Nastaví konfiguraci položky mikrokroků	$CMS[2h][8h]^	mstep_id, speed	[.|!]CMS^
configure current entry	CCT	Nastaví konfiguraci položky proudu	$CCT[2h][2h]^	current_id, current_value	[.|!]CCT^
configure min alt	CMA	Nastaví minimální povolenou výšku nad obzorem	$CMA[1d.6d]^	min_alt	[.|!]CMA^
configure mount type	CMT	Nastaví typ montáže	$CMT[2h]^	mount_type	[.|!]CMT^
configure max meridian dist	CMD	Nastaví maximální vzdálenost přechodu za meridián	$CMD[1d.6d]^	distance	[.|!]CMD^
configure pole crossing	CPC	Nastaví povolení přechodu přes pól při volném pohybu	$CPC[2h]^	crossing	[.|!]CPC^

Příloha B

Dokumentace HW pro FPGA

B.1 Nejvyšší entita

Tabulka B.1: Rozhraní nejvyšší entity

Signál	Směr	Datový typ
IN_RESET	in	std_logic
IN_OSC	in	std_logic
IN_EXEC	in	std_logic
OUT_RDY	out	std_logic
INOUT_SCL	inout	std_logic
INOUT_SDA	inout	std_logic
IN_RA_PENCODER	in	std_logic_vector(1 downto 0)
IN_RA_ZPSWITCH	in	std_logic
IN_RA_AGD_P	in	std_logic
IN_RA_AGD_M	in	std_logic
OUT_RA_STEP	out	std_logic
OUT_RA_DIR	out	std_logic
OUT_RA_MSTEPS	out	std_logic_vector(2 downto 0)
OUT_RA_EN	out	std_logic
OUT_FRA_STEP	out	std_logic
OUT_FRA_DIR	out	std_logic
OUT_FRA_EN	out	std_logic
IN_DEC_PENCODER	in	std_logic_vector(1 downto 0)
IN_DEC_ZPSWITCH	in	std_logic
IN_DEC_AGD_P	in	std_logic
IN_DEC_AGD_M	in	std_logic
OUT_DEC_STEP	out	std_logic
OUT_DEC_DIR	out	std_logic
OUT_DEC_MSTEPS	out	std_logic_vector(2 downto 0)
OUT_DEC_EN	out	std_logic
OUT_FDEC_STEP	out	std_logic
OUT_FDEC_DIR	out	std_logic
OUT_FDEC_EN	out	std_logic

B.2 Adresovatelný registr

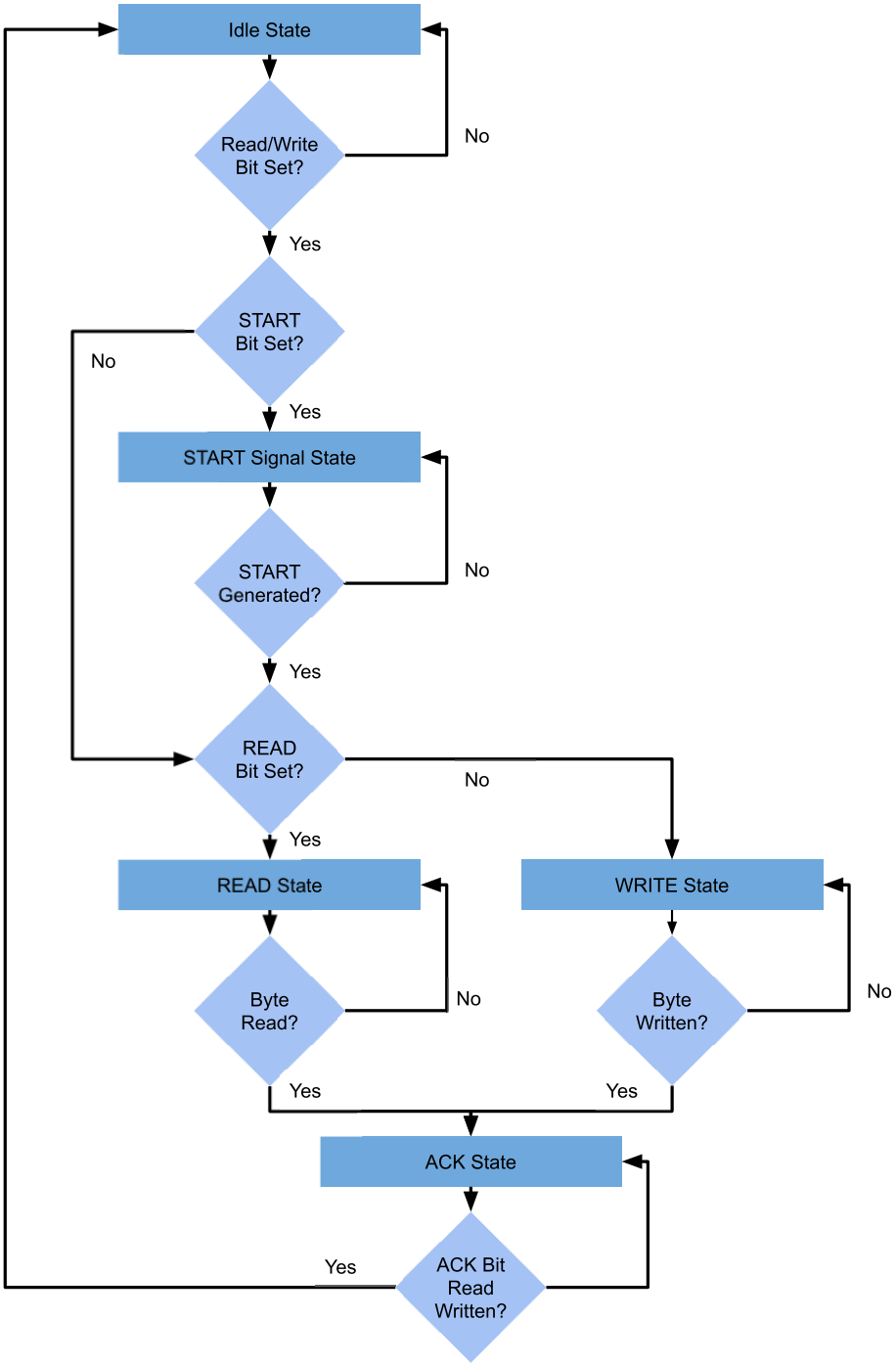
Tabulka B.2: Rozhraní adresovatelného registru

Signál	Směr	Datový typ
IN_CLK	in	std_logic
IN_RST	in	std_logic
IN_EN	in	std_logic
IN_ADDR	in	std_logic_vector(7 downto 0)
IN_DATA	in	std_logic_vector(7 downto 0)
OUT_Q	buffer	std_logic_vector(7 downto 0)

B.3 Wisbone master modul

Tabulka B.3: Rozhraní Wishbone master modulu

Signál	Směr	Datový typ
WB_CLK_I	in	std_logic
WB_DAT_I	in	std_logic_vector(7 downto 0)
WB_DAT_O	out	std_logic_vector(7 downto 0)
WB_RST_I	in	std_logic
WB_TGD_I	in	std_logic_vector(7 downto 0)
WB_TGD_O	out	std_logic_vector(7 downto 0)
WB_ACK_I	in	std_logic
WB_ADR_O	out	std_logic_vector(7 downto 0)
WB_CYC_O	out	std_logic
WB_STALL_I	in	std_logic
WB_ERR_I	in	std_logic
WB_LOCK_O	out	std_logic
WB_RTY_I	in	std_logic
WB_SEL_O	out	std_logic_vector(3 downto 0)
WB_STB_O	buffer	std_logic
WB_TGA_O	out	std_logic_vector(7 downto 0)
WB_TGC_O	out	std_logic_vector(7 downto 0)
WB_WE_O	out	std_logic
MEM_WE_O	out	std_logic
MEM_ADR_O	out	std_logic_vector(7 downto 0)
MEM_DAT_O	out	std_logic_vector(7 downto 0)
MEM_DAT_I	in	std_logic_vector(7 downto 0)



Obrázek B.1: Graf stavového automatu I2C master modulu, graf převzat z [4]

B.4 Entita ovládání krokových motorů

Tabulka B.4: Rozhraní entity pro řízení krokových motorů

Signál	Směr	Datový typ
IN_CLK	in	std_logic
IN_RST	in	std_logic
IN_PGD_CNT	in	std_logic_vector(31 downto 0)
IN_PGD_EN	in	std_logic
OUT_PGD_CNT	out	std_logic_vector(31 downto 0)
IN_AGD_EN	in	std_logic
IN_AGD_DIR	in	std_logic
IN_TRACK	in	std_logic
IN_PERIOD_TRACK	in	std_logic_vector(31 downto 0)
IN_PERIOD_GUIDE	in	std_logic_vector(31 downto 0)
IN_GOTO_CNT	in	std_logic_vector(31 downto 0)
IN_GOTO_EN	in	std_logic
IN_GOTO_CLR	in	std_logic
IN_MOV	in	std_logic
IN_SPID_MOVE	in	std_logic_vector(15 downto 0)
IN_SPID_GOTO	in	std_logic_vector(15 downto 0)
OUT_MEM_ADDR	buffer	std_logic_vector(9 downto 0)
IN_MEM_PERIOD	in	std_logic_vector(31 downto 0)
OUT_DIR	buffer	std_logic
OUT_STEP	buffer	std_logic
OUT_MSTEPS	buffer	std_logic_vector(2 downto 0)
IN_SYN	in	std_logic
IN_DIR	in	std_logic
OUT_MAIN_CNT	buffer	std_logic_vector(31 downto 0)

Příloha C

Dokumentace kódu MCU

Kód mikrokontroléru je postaven na knihovnách dostupných skrze Advanced Software Framework 4 (dále jen ASF). Je dělen podle jednotlivých funkčních bloků do adresářů obsahujících soubory řešící danou problematiku. Veškerý zdrojový kód mikrokontroléru se nachází v adresáři *./src/mcu*, který je dále pro potřeby dokumentace nazýván kořenovým adresářem. Zde Atmel studio zakládá svůj projekt a všechny zdrojové soubory, soubory potřebné pro překlad a všechny výstupy překladu, jako binární paměťové soubory, jsou umístěny zde.

Složka `./Debug` obsahuje automaticky generovaný Makefile a také všechny výstupy překladu. Podadresář `./Debug/src` zrcadlí strukturu adresáře a do odpovídajících složek ukládá objektové soubory, které jsou výstupy překladu odpovídajících souborů zdrojového kódu. Kompletní binární soubory k naprogramování do MCU jsou umístěny v kořeni adresáře `Debug`.

Složka `./src` obsahuje veškerý zdrojový kód mikrokontroléru jednotky.

V podsložce `./src/ASF` jsou uloženy veškeré knihovny importované pomocí ASF. Tento obsah je automatický generovaný pomocí nástroje *ASF Wizard*, který je součástí vývojového prostředí Atmel Studio. Jedinou výjimku tvoří soubory v adresáři `./src/ASF/common/boards/user_board`. Ten obsahuje soubory s definicemi konstant vázaných na konkrétní desku a funkce pro inicializaci hardware této desky.

Podsložka `./src/components` obsahuje většinu autorské práce z kódu MCU. Obsah adresáře je členěn do složek podle problematiky, kterou daný kód řeší, například funkce pro astronomické výpočty jsou obsaženy v souborech ve složce `amath`, soubory s funkcemi provádějícími přijímaní, zpracování a odpovědi na příkazy jsou ve složce `commands`, apod. V kořeni složky `./src/components` je navíc umístěn důležitý soubor `common.h`. Tento soubor obsahuje definice datových struktur využívaných v téměř celém kódu programu mikrokontroléru.

C.1 Hlavní soubor

Přímo v kořenovém adresáři zdrojového kódu je umístěn soubor s hlavní funkcí programu MCU. Ta představuje vstupní bod celého programu. Definuje globální proměnné a importuje soubory a knihovny. Podrobný popis průběhu hlavní smyčky je uveden v podkapitole 5.2.

C.2 Astronomická matematika

Vlastní implementace optimalizovaných matematických funkcích používaných ve výpočtech MCU. Je uložena ve složce *./src/components/amath* v následujících souborech:

- `astronomy_math.c`
- `astronomy_math.h`

Hlavičkový soubor obsahuje prototypy funkci a konstanty před-počítané při překladu pro optimalizaci náročných výpočtů na zařízeních s omezeným výkonem.

```
#define PI 3.14159265358979323846
#define TWO_PI (PI + PI)
#define RD (360 / (PI * 2))
#define DR ((PI * 2) / 360)
#define JD2000 2451545.0 // julianske datum dne 1.1.2000, 12:00 UT
#define JULIAN_YEAR 365.25
#define JULIAN_CENTURY 36525.0
#define GREGORIAN_YEAR 365.2425
#define GREGORIAN_CENTURY 36524.25
#define INV_DAY_SECONDS (1 / 86400.0)
```

Využití těchto konstant ušetří důležitý výpočetní čas na zařízeních, které nemají FPU a musí dělení nahrazovat sérií procesorových instrukcí. Konstanty jsou velmi často využívané při výpočtech převodu mezi radiány a stupni, při převodech mezi horizontálním a rovníkovým souřadným systémem a při přepočtu data na juliánské datum a zpět. Všechny tyto funkce, a další optimalizované výpočty řeší tato knihovna.

```
int32_t _ipow(int32_t base, int32_t exp)
```

Tato funkce provádí optimalizovaný výpočet mocniny ze základu celého čísla umocněného na exponent. pro výpočet je použit pouze bitový operátor AND, násobení a bitový posun vpravo.

```
double_t _fracf(double_t input)
```

Funkce vrátí desetinnou část čísla v plovoucí řádové čárce. jedná se pouze o obalující funkci nad knihovní funkcí `modf()`, která zahazuje celou část čísla, kterou knihovní funkce vrací.

```
uint32_t _rotl(uint32_t value, uint8_t shift)
uint32_t _rotr(uint32_t value, uint8_t shift)
```

Tyto dvě funkce jsou implementací rychlé bitové rotace v C za pomoci celočíselného násobení a bitových posunů.

```
double_vec2_t convert_azalt_to_radec(double_vec2_t hz_coords, double_t jd)
double_vec2_t convert_radec_to_azalt(double_vec2_t eq_coords, double_t jd)
```

Tato dvojice inverzních funkcí slouží k převodu mezi jednotlivými souřadnými systémy.

Funkce je časově poměrně náročná. Pro potřebu jednotky je však dostačující. Převod se provádí pouze v situacích, kdy dalekohled musí najet do pozice zadané v horizontálním souřadném systému, či z této pozice najet na rovníkové souřadnice.

Pro zrychlení výpočtu se používají předem vypočítané hodnoty normálně získávané náročnými operacemi. Tyto hodnoty mohou být jednak hodnoty získané při překladu (`PI`,

TWO_PI, konstanty pro převod mezi radiány a stupni) nebo hodnoty získané pouze při aktualizaci některých konfiguračních dat. Příkladem může být hodnota sinu a cosinu zeměpisné šířky, která se přepočítá pouze jednou při aktualizaci nastavení zeměpisných souřadnic.

```
uint32_t convert_rads_to_steps(double_t angle, motor_axis_t axis)
double_t convert_steps_to_rads(uint32_t steps, motor_axis_t axis)
```

Poměrně jednoduché funkce jsou nezbytné pro komunikaci s FPGA. Hradlové pole pracuje s celými čísly uloženými v čítačích, potřebuje proto převést před zapsáním informaci o úhlu do počtu kroků jednotlivých motorů. Informace o ose je nezbytná pro určení, který počet kroků motoru na otáčku uložený v konfiguraci je třeba použít. U větším montáží není výjimečné, že se počet kroků na otáčku mezi osami liší.

```
void convert_jd_to_time(double_t jd, twi_rtc_payload_t * time)
void convert_time_to_jd(double_t * jd, twi_rtc_payload_t * time)
```

Pár optimalizovaných funkcí pro převod mezi datovou strukturou ukládající čas a juliánským datem. Datum a čas potřebné pro jednotlivé výpočty se uchovává jako juliánské datum ve stavu jednotky. Čítač sekund řízený přerušením generovaným RTC modulem poskytuje dostatečnou přesnost pro výpočty po dobu, co je jednotka spuštěna. Po zapnutí a při příkazu synchronizaci je potřeba čas uložený ve stavu aktualizovat z času udržovaného v RTC modulu. RTC modul má registry obsahující datum a čas přístupné přes I2C, ovšem uložené ve standardním formátu `[rok, měsíc, den, hodina, minuta, sekunda]`. Tento formát je při přečtení potřeba přepočítat do juliánského data pro použití ve výpočtech a při nastavování času přepočítat juliánské datum do tohoto formátu.

C.3 Zpracování příkazů

Funkce pro příjímání, zpracování a odpovídání na příkazy z PC nebo ruční ovládací jednotky. Jsou uloženy ve složce `./src/components/commands` v následujících souborech

- `command_parser.c`
- `command_parser.h`
- `commands.h`
- `config_commands.c`
- `config_commands.`
- `protocol_commands.c`
- `protocol_commands.h`
- `replies.c`
- `replies.h`
- `responder.c`
- `responder.h`

C.3.1 Definice příkazů

Zpracování příkazů se provádí z bufferu přijatých dat. Ten může pocházet ze kteréhokoli zdroje - USB, Ethernet či RS232 - a je předán funkcím ke zpracování. To vytváří mezivrstvu

při zpracování komunikace a umožňuje propojit snadno univerzální funkce se specializovaným jednotlivých metod komunikace.

Nejpodstatnějším souborem je hlavičkový soubor commands.h. Ten obsahuje definici a vlastnosti všech příkazů, kterým jednotka rozumí.

Základem je enumerační typ `command_t`, který obsahuje výčet všech příkazů. Hodnotu tohoto typu vrací funkce vyčítající typ příkazu z příchozího bufferu.

Hodnota typu `command_origin_t` je nastavena v hlavní smyčce při přijetí příkazu. Tento mechanismus zaručuje odeslání odpovědí po stejném komunikačním kanálu, po kterém byl příkaz přijat. Při zpracování je podstatná série konstant `<COMMAND>_INDEX`. Tyto konstanty definují indexy jednotlivých příkazů pro dekódování příkazu. Konstanty indexů se vypočítají pří překladu jako

```
(('{0}' << 16) + ('{1}' << 8) + '{2}')
```

kde `{0}` představuje první znak z kódu příkazu, `{1}` druhý znak a `{3}` třetí znak kódu příkazu. Převedením tří libovolných znaků na jediné celé číslo unikátní pro každou kombinaci umožňuje rychlé zpracování kteréhokoli kódu pomocí jediného skoku.

Konstanty `<COMMAND>_LENGTH` slouží pro kontrolu správné délky přijatého příkazu. Funkce pro získání příkazu porovná délku přijatého bufferu na odpovídající konstantu.

C.3.2 Zpracování přijatého příkazu

```
command_t parse_command_from_buffer(const char * buffer, int length)
```

Funkce sloužící pro určení příkazu přijatého v bufferu. Buffer není při zpracování upravován. Funkce přečte počáteční tři znaky a na jejich základě vrátí odpovídající příkaz. Pokud je hodnota parametru `length` 0, funkce neprovádí kontrolu délky. V ostatních případech porovná před navrácením nalezeného příkazu velikost parametru `length` na konstantu `<COMMAND>_LENGTH` odpovídajícího příkazu. Pokud délka neodpovídá, funkce vrátí `C_CORRUPT` namísto příkazu. Pokud přijatý příkaz neodpovídá žádnému ze známých příkazů, funkce vrátí `C_UNKNOWN`.

Kontrola délky není podstatná při zpracování příkazů přijatých přes USB, či Ethernet. V těchto případech kontrolu na chyby provádí nižší vrstva a není potřeba chyby kontrolovat manuálně.

```
int parse_uint32_from_char_base16(const char * buffer, int offset,
                                  uint32_t * result)
int parse_uint16_from_char_base16(const char * buffer, int offset,
                                  uint16_t * result)
int parse_uint8_from_char_base16(const char * buffer, int offset,
                                 uint8_t * result)
int parse_int16_from_char_base16(const char * buffer, int offset,
                                 int16_t * result)
```

Sada funkcí pro vyčtení různých typů celých čísel z přijatého bufferu. Parametry příkazů jsou jednotce předávány jako řetězce znaků reprezentující čísla v hexadecimálním formátu. Parametr `buffer` obsahuje ukazatel na pole přijatých znaků, ze kterých bude hodnota načtena. Parametr `offset` udává pozici prvního znaku čísla vůči počátku bufferu. Parametr `result` obsahuje ukazatel na celé číslo v paměti, kam bude přečtené číslo uloženo.

Pokud přečtení proběhne úspěšně, funkce navrací konstantu `PARSER_OK`. Pokud je v bufferu neznámý znak (jiný, než `[0-9a-fA-F]`), funkce navrací vrátí `PARSER_ERROR`.

```
int parse_angle_from_char_base10(const char * buffer, int offset,
                                 double_t * result)
```

Funkce pro vyčtení úhlu (číslo v plovoucí řádové čárce) z přijatého bufferu. Úhel je vždy kladný a v bufferu je reprezentován řetězcem znaků ve formátu odpovídajícím regulárnímu výrazu `[0-9]\.[0-9]{6}`. Parametr `buffer` obsahuje ukazatel na pole znaků, ze kterých bude úhel načten. Parametr `offset` udává pozici prvního znaku úhlu vůči počátku bufferu. Parametr `result` obsahuje ukazatel na číslo v plovoucí řádové čárce v paměti, kam bude úhel uložen.

```
int parse_s_angle_from_char_base10(const char * buffer, int offset,
                                   double_t * result)
```

Tato funkce funguje identicky jako funkce `parse_angle_from_char_base10`, ovšem předpokládá, že řetězec bufferu je předcházen znakem znaménka + nebo -. Pokud je úhel předcházen znakem -, hodnota uložená na pozici `result` je znegována.

```
int parse_jd_from_char_base10(const char * buffer, int offset,
                              double_t * result)
```

Funkce pro vyčtení juliánského data (čísla v plovoucí řádové čárce) z přijatého bufferu. Juliánské datum je vždy kladné číslo a v bufferu je reprezentováno řetězcem znaků ve formátu odpovídajícím regulárnímu výrazu `[0-9]{7}\.[0-9]{8}`. Parametr `buffer` obsahuje ukazatel na pole znaků, ze kterých bude úhel načten. Parametr `offset` udává pozici prvního znaku úhlu vůči počátku bufferu. Parametr `result` obsahuje ukazatel na číslo v plovoucí řádové čárce v paměti, kam bude juliánské datum uloženo.

C.3.3 Provedení příkazu protokolu

Provádění příkazů zajišťuje sada funkcí, každá vykonávající právě jeden z příkazů kterým jednotka rozumí. Tyto příkazy jsou volány z hlavní smyčky po dekódování přijatého příkazu. Veškeré načtení parametrů z bufferu, provedení potřebných operací a sestavení odpovědí se provádí v těchto funkcích. Pokud příkaz požaduje parametry, globální vstupní buffer je předán jako parametr funkci.

Společnou vlastností naprosté většiny těchto funkcí (mimo ty, které nemohou způsobit chybu) je návěští `FAILURE:` umístěné za příkazem `return`. Za tímto návěštím je obsažen kód pro úpravu pozitivní odpovědi na negativní a změna globální proměnné `response_length`, kterou kód odesílající odpověď čte. Na toto návěští se provádí skok v rámci funkce v případě, že některá z funkcí volaných uvnitř vrátí chybu a provádění příkazu se stane nemožným. Tím může být chyba při čtení parametrů z bufferu (neznámý znak), či chyba při komunikaci s FPGA nebo RTC. V tom druhém případě jde o kritickou hardware chybu a jednotku je potřeba zcela resetovat. Toho je docíleno nastavením globálního příznaku `critical_failure` na `true`, který je testován na konci hlavní smyčky a zastaví její vykonávání.

Pokud některá funkce mění stav jednotky, jsou proměnné uvnitř funkce inicializovány na aktuální stav. Veškeré uložení nových údajů do paměti se provádí poté, co jsou všechny rizikové operace funkce úspěšně dokončeny. To zahrnuje jak zpracování parametrů, tak komunikaci s FPGA, i když takovéto selhání je považováno za chybu, ze které se nelze zotavit a konzistence stavových dat není poté podstatná.

```
void command_no_operation(void)
```

Základní funkce. Její veškerá funkčnost spočívá ve sestavení pozitivní odpovědi. Sama o sobě neprovádí žádnou operaci a nemůže vrátit chybu.

`void command_get_version(void)`

Funkce vrací verzi firmware řídící jednotky ve formátu `major.minor.build.revision`. Funkce nemůže vrátit chybu.

`void command_select_mov_speed(const char * buffer)`

Funkce sloužící k nastavení rychlosti používané při příkazu MOV (viz A.1). Po úspěšném nastavení nastaví globální příznak `move_speed_updated` na `true`. Tímto příznakem se řídí funkce měnící stav volného pohybu v FPGA.

`void command_start_move(const char * buffer)`

Funkce nastaví směr přijatý v parametru příkazu jako aktivní směr volného pohybu v FPGA a krátce nastaví `FPGA_execute` signál na log. 1. Pokud je příznak `move_speed_updated` nastaven na `true`, funkce nejprve zapíše to FPGA novou cílovou rychlost volného pohybu.

`void command_stop_move(const char * buffer)`

Funkce nastaví směr přijatý v parametru příkazu jako neaktivní směr volného pohybu v FPGA a krátce nastaví `FPGA_execute` signál na log. 1.

`void command_goto(const char * buffer)`

Funkce provede přesun na definované souřadnice, pro teoretický popis viz 3.5.5.

Funkce spočítá rozdíl současných a cílových souřadnic, převede jej na počet kroků motoru a tyto hodnoty zapíše do FPGA. Současné souřadnice jednotka získá jako referenční souřadnice sečtené počtem kroků v čítačích pohybu v FPGA převedené na úhel. Rektascenzi dodatečně upraví o úhel uražený za čas od okamžiku poslední synchronizace.

`void command_stop_goto(void)`

Funkce zapíše do FPGA příkaz pro vynulování interních čítačů přesunu na souřadnice a krátce nastaví `FPGA_execute` signál na log. 1.

`void command_synchronize(const char * buffer)`

Funkce provede synchronizaci, pro teoretický popis viz 3.5.1.

Funkce přečte parametry příkazu a uloží je do paměti jednotky jako referenční souřadnice. Následně zapíše do FPGA příkaz pro synchronizaci. Poté přečte z RTC modulu aktuální čas, který také uloží do paměti jednotky. Nakonec funkce vynuluje čítač sekund `timer_sec` a krátce nastaví `FPGA_execute` signál na log. 1. Ten způsobí provedení zapsaného příkazu v FPGA při kterém dojde k vynulování interních čítačů kroků.

`void command_flip_pier_side(void)`

Funkce provede přeložení montáže, pro teoretický popis viz 3.5.6.

Funkce překlopí dalekohled na druhou strany pilíře. Fungování je velmi podobné příkazu přesunu na souřadnice, ale pohybuje dalekohledem o předem známé počty kroků. Z aktuálních souřadnic a stavové informaci o straně od pilíře spočítá pozice krokových motorů se stejnými souřadnicemi na druhé straně od pilíře. Rozdíl počtu kroků zapíše do FPGA a předá příkaz k pohybu.

`void command_set_julian_date(const char * buffer)`

Funkce slouží k aktualizaci stavové informace RTC modulu na desce. Přijímá juliánské datum jako parametr příkazu, které převede na časovou strukturu a tu zapíše po I2C do RTC modulu.

`void command_set_location(const char * buffer)`

Funkce aktualizuje stavové informace o geografických souřadnicích dalekohledu. Přijímá souřadnice jako parametr příkazu. Z těchto souřadnic vypočítá také sinus a cosinus zeměpisné šířky, které zapíše do paměti vede samotných souřadnic.

`void command_get_state(void)`

Funkce vrátí v odpovědi stav jednotky. Stav jednotky má velikost 64 bitů a je složen z položek Stav (16b), Typ montáže (8b), Stav sledování (8b), Pozice od pilíře (8b), Zemská polokoule (8b) a Příznaky aktualizování stavu jednotky (16b). Funkce nemůže vrátit chybu.

`void command_get_ra_dec(void)`

Funkce vrátí nebeské souřadnice dalekohledu. Funkce přečte aktuální hodnotu z čítačů pozice z FPGA (viz 4.5.9), spočítá současné souřadnice a ty vrátí jako dva úhly v odpovědi. Funkce využívá funkcí z matematické knihovny C.2.

`void command_get_location(void)`

Funkce vrátí zeměpisné souřadnice zapsané v paměti jednotky. Funkce nemůže vrátit chybu.

`void command_get_julian_date(void)`

Funkce načte časovou strukturu z RTC modulu po I2C, převede ji na juliánské datum a vrátí v parametru odpovědi.

`void command_set_tracking(const char * buffer)`

Funkce nastaví režim sledování jednotky podle hodnoty parametru příjatého v příkazu. Návaznost režimů sledování na příchozí hodnoty je popsána v následující tabulce.

Tabulka C.1: Hodnoty režimů sledování

Hodnota	Režim sledování
1	Sledování vypnuto
2	Sledování hvězdnou rychlostí
3	Sledování Měsíční rychlostí
4	Sledování Sluneční rychlostí
5	Sledování Královou rychlostí[1]
6	Sledování uživatelskou rychlostí

`void command_set_hemisphere(const char * buffer)`

Funkce nastaví zemskou polokouli jednotky podle hodnoty parametru přijatého v příkazu.

`void command_set_pier_side(const char * buffer)`

Funkce nastaví stranu od pilíře jednotky podle hodnoty parametru přijatého v příkazu ale pouze pokud typ montáže není nastaven jako vidlicový.

`void command_pulse_guide(const char * buffer)`

Funkce zapíše směr a dobu pro aktivitu pulzního korekce do FPGA. Absolutní hodnoty přijatých parametrů v milisekundách převede na počet taktů FPGA. Pokud byl některý z časových parametrů záporný, funkce nastaví nejvyšší bit odpovídajícího počtu taktů. Pokud byl parametr kladný, funkce tento bit vynuluje.

[1] Králova rychlost je přítomna pro potřeby budoucích rozšíření

`void command_set_park(void)`

Funkce zapíše aktuální nebeské souřadnice dalekohledu do paměti jednotky ve formátu horizontálních souřadnic jako parkovací souřadnice.

`void command_park(void)`

Funkce zapíše aktuální hodnotu nebeských souřadnic dalekohledu do paměti a nasměruje dalekohled na pozici určenou parkovacími souřadnicemi. Zapíše aktuální nastavení sledování do paměti jako aktivitu před zaparkováním a zastaví všechny druhy pohybu.

`void command_unpark(void)`

Funkce nasměruje dalekohled na souřadnice zapsané při posledním příkazu parkování. Pokud byla aktivní hodnota sledování při přijetí posledního příkazu k parkování, jednotka tento typ sledování opětovně aktivuje.

`void command_abort(void)`

Funkce zápisem do FPGA okamžitě zastaví všechny druhy pohybu. Následné provedení operace obnoví všechny aktivity pohybu na výchozí, tedy vypnuté, hodnoty

C.3.4 Nastavení konfigurace

Tato skupina funkcí je zodpovědná za správu a modifikaci konfiguračních a stavových hodnot jednotky. Hlavní charakteristikou je, že tyto funkce nejsou součástí základního komunikačního protokolu. Ovladač pro komunikaci s jednotkou je nepodporuje. Tyto funkce jde také jako jediné volat i v neinicializované jednotce.

`void unit_init(void)`

Funkce inicializuje stavovou paměť MCU. Využívá k tomu hodnot uložených v konfiguračních datech jednotky.

```
void command_configure_motor_entry(const char * buffer)
void command_configure_encoder_entry(const char * buffer)
```

Funkce nastaví konfiguraci motoru, či enkodéru v ose specifikované parametrem příkazu na hodnoty specifikované v dalších parametrech. Následně zapíše odpovídající konfiguraci do FPGA. Konfigurace pro motory a enkodéry v FPGA je uložena každá v jediném bytu. Modifikace kteréhokoli motoru či enkodéru aktualizuje obě hodnoty v FPGA, ale pouze změněná hodnota se liší.

```
void command_configure_speed_entry(const char * buffer)
void command_configure_mstep_entry(const char * buffer)
```

Funkce sloužící pro nastavení parametrů, podle kterých se generuje tabulka rychlostí pro nahrání do FPGA. V současném řešení je generování i nahrání přesunuto z kódu MCU do samostatného skriptu a kódu FPGA. Funkčnost těchto funkcí je založena na starém návrhu.

`void command_configure_current_entry(const char * buffer)`

Funkce slouží k nastavení proudu vedeného do krokových motorů. Proud je převeden na odpovídající napětí, které je následně nahráno do registru DAC na plošném spoji. Ten změní napětí na referenčním vstupu řadiče krokových motorů, který změní proud tekoucí do motorů.

`void command_configure_min_alt(const char * buffer)`

Funkce slouží k nastavení hodnoty minimální výšky nad obzorem uložené v konfiguraci jednotky.

```
void command_configure_mount_type(const char * buffer)
```

Funkce slouží k nastavení typu montáže uloženého v konfiguraci jednotky.

```
void command_configure_max_meridian_dist(const char * buffer)
```

Funkce slouží k nastavení maximální vzdálenosti za meridiánem uloženéh v konfiguraci jednotky, kterou může jednotka urazit, než je nutné překlopení dalekohledu na druhou stranu od pilíře montáže.

```
void command_configure_pole_crossing(const char * buffer)
```

Funkce nastavuje příznak v konfiguraci jednotky, který umožňuje, nebo zakazuje přechod přes pól při přesunu na souřadnice.

C.3.5 Sestavení obecných odpovědí

Tyto soubory obsahují sadu bez parametrů a návratových hodnot funkcí pro sestavení odpovědí do globálního bufferu. Odeslání odpovědi se řídí délkou odpovědi uloženou v globální proměnné response_length, která je těmito funkcemi nastavována. Technicky tak dojde ke kompletnímu přepsání odpovědi z pohledu odesílání.

```
void prepare_nack_reply(void)
void prepare_ucm_reply(void)
void prepare_ovr_reply(void)
```

Funkce sestavující odpovědi podle standardního formátu definovaného protokolem.

C.3.6 Sestavení obsahu odpovědí

Funkce ze souboru `responder.c` jsou v podstatě opakem dříve popsaných funkcí pro vyčítání parametrů - ovšem namísto čtení znaků z přijatého bufferu a následného interpretování těchto znaků jako číslo, berou tyto funkce vstupní hodnotu a generují řetězec znaků pro odeslání.

```
void assemble_char_base16_from_uint8(char * buffer, int offset,
                                     uint8_t integer)
void assemble_char_base16_from_uint16(char * buffer, int offset,
                                     uint16_t integer)
void assemble_char_base16_from_uint32(char * buffer, int offset,
                                     uint32_t integer)
```

Tyto funkce přijmou celočíselný parametr integer a po čtveřicích bitů jej převedou na hexadecimální znaky které uloží na odpovídající pozice pole znaků, jehož adresa je předána v parametru buffer.

```
void assemble_char_angle_from_double(char * buffer, int offset,
                                     double_t angle)
void assemble_char_s_angle_from_double(char * buffer, int offset,
                                       double_t angle)
```

Tyto funkce přijímají parametr v plovoucí řádové čárce s dvojitou přesností, který následně převedou na řetězec znaků reprezentující přijatý parametr v desítkové soustavě. Číslo je reprezentováno, jako jeho absolutní hodnota.

Funkce `assemble_char_s_angle_from_double` zapíše do bufferu před výsledek znak znaménko podle znaménka přijatého parametru.

```
void assemble_char_jd_from_double(char * buffer, int offset, double_t jd)
```

Tato funkce přijme parametr v plovoucí řádové čárce s dvojitou přesností, který následně převede na řetězec znaků reprezentující juliánské datum v desítkové soustavě. Číslo je reprezentováno, jako jeho absolutní hodnota.

C.4 Výchozí konfigurace

Pomocné funkce pro operace na konfigurací jednotky. Hlavním účelem je inicializace na výchozí hodnoty a aktualizace hodnot. Jsou uloženy ve složce `{./src/components/conf` v následujících souborech

- config_utils.c
- config_utils.h

```
void init_defaults(void)
```

Funkce je volána v případě chyby načtení dat z paměti, nebo v případě, že data v paměti nejsou uložena. Nakonfiguruje jednotku na základní hodnoty nutné pro spuštění, ale ruční konfigurace je nutná, aby jednotka začala přijímat příkazy.

Ve vývojové verzi programu používaném v jednotce je toto omezení zrušeno - jednotka není připojena k montáži, nebo je velmi úzce kontrolována a nehrozí škoda na majetku.

```
int update_period_entry(uint8_t id)
void update_periods_config(void)
```

Dvojice funkce pro automatickou aktualizaci tabulky period, která je v paměti MCU uložená vedle tabulky rychlostí. Položku tabulky period je potřeba přepočítat vždy při aktualizaci některé z rychlostí.

Funkce `update_periods_config` slouží pro automatické přepočítání všech položek a je využívána k naplnění tabulky po inicializaci na výchozí hodnoty.

```
uint16_t get_speed_index(uint32_t period, motor_axis_t axis)
```

Funkce sloužící k nalezení indexu nejbližší periody v předem vypočítané tabulce period nahrávané do FPGA. Tabulka period v FPGA obsahuje pouze 1024 položek. Při zadání nové požadované rychlosti je ji potřeba převést na index nejbližší možné rychlosti v této tabulce. Tato tabulka je v MCU uložena ve samostatném konstantním poli pro každou osu.

```
int update_speed_id_entry(uint8_t id)
void update_speed_ids_config(void)
```

Dvojice funkce pro automatickou aktualizaci záznamů o indexech v předem vypočítané tabulce period, která je nahrávána do RAM v FPGA. Indexy položek tabulky je potřeba přepočítat pokaždé při aktualizaci nastavení rychlosti. Pro zrychlení operace lze využit znalosti, že počáteční rychlost rampy `guide` je vždy první položkou v tabulce a rychlost `slew` je vždy poslední rychlostí. ID prostřední rychlosti `center` je potřeba vypočítat

Funkce `update_speed_ids_config` slouží pro automatické přepočítání všech položek a je využívána k naplnění tabulky indexů po inicializaci na výchozí hodnoty.

C.5 Ovládání DAC

Funkce pro ovládání zápisu do registru digitálně analogového převodníku3.2.6 pomocí GPIO rozhraní v MCU. Jsou uloženy ve složce
`./src/components/dac` v následujících souborech

- `config_utils.c`
- `config_utils.h`

Popis fungování digitálně-analogového převodníku je popsán v kapitole 3.2.6. V jednotce není horní bit ze dvou bitů určujících výstup nikdy nastavován a má stálou hodnotu 0. Dolní z těchto bitů má hodnotu podle osy, jejíž proud do motorů má být aktualizován. Pro osu rektascenze má hodnotu 1, pro osu deklinace 0. *RNG* bit má konstantní hodnotu 0. Hodnota *CODE* je závislá na konfiguraci jednotku určované uživatelem.

```
int set_current(uint32_t current)
```

Funkce provede výše zmíněnou sekvenci změn na GPIO signálech pro nahrání parametru do registru DAC. Parametrem je již 11 bitů.

```
int DAC_set_current_standby(motor_axis_t motor)
int DAC_set_current_tracking(motor_axis_t motor)
int DAC_set_current_slew(motor_axis_t motor)
```

Funkce pro nahrání hodnoty uložené v konfiguračních datech do kontrolního registru DAC. Parametr osy určuje hodnoty trojice kontrolních bitů, samotná funkce pak hodnotu, která bude nahrána.

C.6 Komunikace s FPGA

Funkce zprostředkovávající veškerou komunikaci s FPGA4. MCU komunikuje s FPGA dvěma způsoby. Tím jednodušším je komunikace přes GPIO vstupy/výstupy MCU. Tahle metoda je optimální pro rychlé předávání jedno-bitových informací signalizaci stavu. Pro přenášení většího množství dat je ale zcela nedostačující a vyžadovala by implementaci komunikačního protokolu. Řešením je již navržená a velmi rozšířená sběrnice I2C. Ta je podporována přímo v hardware MCU za pomoci ASF knihoven, kterých je kódu využíváno. Zdrojový kód je uložen ve složce `./src/components/fpga` v následujících souborech

- `fpga_commands.c`
- `fpga_commands.h`
- `fpga_wrapper.c`
- `fpga_wrapper.h`
- `fpga.h`

Kód je rozdělen do dvou párů souborů podle úrovně na které operuje. Funkce v souboru `fpga_wrapper.c` pracují na nízké úrovni - ať už mění či čtou signály na GPIO výstupech a výstupech, nebo využívají ASF funkcí pro I2C komunikaci. Funkce definované v `fpga_commands.c` poté zabalují funkce na nižší úrovni pro snadnější použití.

C.6.1 Definice konstant a struktur

Definice konstant a datových struktur potřebných po komunikaci s FPGA je vyčleněna mimo soubory s funkcemi do samostatného hlavičkového souboru `fpga.h`. Ten obsahuje definice kódů jednotlivých příkazů zasílaných do FPGA, bitových masek, adres paměťového prostoru a struktur potřebných pro komunikaci. Specifikem struktur pro komunikaci je zarovnání na 1 byte pro přenos po I2C.

C.6.2 Komunikace na nízké úrovni

```
int FPGA_is_busy(void)
```

Funkce vrátí `true`, pokud je signál signalizující zaneprázdněnost FPGA na log. 1. V opačném případě vrátí `false`.

```
void FPGA_execute(void)
```

Funkce krátce nastaví signál provedení příkazu v FPGA na log. 1 a poté opět na log. 0.

```
void FPGA_reset(void)
```

Funkce krátce nastaví signál resetování FPGA na log. 1 a poté opět na log. 0.

```
int FPGA_read(unsigned char addr, char * buffer, unsigned char size)
int FPGA_write(unsigned char addr, char * buffer, unsigned char size)
```

Dvojice funkcí pro komunikaci s FPGA. Funkčnost je založena na ASF a využívá funkcí pro čtení a zápis po I2C. ASF také definuje strukturu zprávy, kterou tyto funkce přijímají. Komunikace je založena na správném zapsání adresy zařízení, ukazatele na místo v paměti, které má být odesláno, či kam má být načten výsledek a velikost tohoto paměťového bloku v bytech do struktury a zavolání odpovídající funkce zápisu nebo čtení s parametrem této struktury.

C.6.3 Komunikace z pohledu volajících funkcí

Tato skupina volá funkce předchozí skupiny s odpovídajícími parametry. To umožňuje zjednodušení volání zápisů a čtení z FPGA v ostatních částech kódu. Veškerá potřebná data jsou předána funkcím jako přirozené parametry a převedeny na odpovídající struktury pro zápis do FPGA uvnitř funkcí.

```
int FPGA_init(void)
```

Funkce nahraje všechna potřebná data z paměti MCU do registrů FPGA.

```
int FPGA_write_command(command_t command)
```

Funkce nahraje příkaz předaný v parametru do odpovídajícího registru v FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_write_move(uint8_t new_ra_mov_state, uint8_t new_dec_mov_state)
```

Funkce zapíše stav volného pohybu složený ze dvou přijatých parametrů do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_goto(uint32_t ra_counter, uint32_t dec_counter,
uint8_t ra_direction, uint8_t dec_direction)
```

Funkce zapíše hodnoty čítačů a směrů pro použití při přesunu na souřadnice složené z přijatých parametrů do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_read_ra_cnt(uint32_t * ra_cnt)
int FPGA_read_dec_cnt(uint32_t * dec_cnt)
```

Funkce přečte hodnotu čítače kroků odpovídající osy z registru FPGA a zapíše ji na pozici danou ukazatelem v parametru. Adresa registru je skryta volající funkci.

```
int FPGA_write_tracking(uint8_t new_tracking)
```

Funkce zapíše aktivitu sledování podle přijatého parametru do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_write_pulse_guide(uint32_t ra_pgd_counter,
                           uint32_t dec_pgd_counter)
```

Funkce zapíše doby trvání pulzního korekce do registrů FPGA. Zapsaná hodnota obsahuje jak počet taktů FPGA, tak i směr pohybu, viz 4.5.1. Adresa registru je skryta volající funkci.

```
int FPGA_write_motor_config(uint8_t new_ra_config,
                            uint8_t new_dec_config)
```

Funkce sestaví nový vektor konfigurace motorů v FPGA z přijatých parametrů a ten zapíše do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_write_encoder_config(uint8_t new_ra_config,
                              uint8_t new_dec_config)
```

Funkce sestaví nový vektor konfigurace enkodérů v FPGA z přijatých parametrů a ten zapíše do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_write_period_track(uint32_t new_period)
int FPGA_write_period_guide(uint32_t new_period)
```

Funkce zapíše do registru FPGA odpovídající periodu rychlosti. Adresa registru je skryta volající funkci.

```
int FPGA_write_spid_move(uint16_t speed_id)
int FPGA_write_spid_goto(uint16_t speed_id)
```

Funkce zapíše do registru FPGA odpovídající ID rampující rychlosti. Adresa registru je skryta volající funkci.

```
int FPGA_write_pier_side(uint8_t new_pier_side)
```

Funkce zapíše bitovou konfiguraci pozice od pilíře do registru FPGA. Adresa registru je skryta volající funkci.

```
int FPGA_write_hemisphere(uint8_t new_hemisphere)
```

Funkce zapíše bitovou konfiguraci zemské polokoule do registru FPGA. Adresa registru je skryta volající funkci.

C.7 Obsluha přerušení

Sada funkcí obsluhujících přerušení. Jsou využívány zpravidla v komunikace po sériové lince, ale také k počítání času či reakci na jiná přerušení po GPIO. Zdrojový kód je uložen ve složce `./src/components/isr` v následujících souborech

- `isr.c`
- `isr.h`

Přerušení jsou časově velmi náročná činnost. Je podstatné, aby funkce obsluhující přerušení prováděly minimální množství operací a pomocí příznaků předávaly informace časově relativně nenáročné hlavní smyčce.

```
void intc_USART0_char_received(void)
void intc_USART1_char_received(void)
```

Funkce jsou vyvolány při přijetí znaku po sériové lince. Jejich činností je načítaní přijatého znaku do příkazového bufferu. Počáteční znak protokolu `$` způsobí vynulování ukazatele do pole znaků sloužícího jako příchozí buffer. Tímto ukazatelem se přerušení řídí a jeho vynulování představuje přepsání bufferu. Ukončovací znak `^` způsobí nastavení příznaku přijatého příkazu. Všechny ostatní znaky jsou vloženy do pole na pozici indexu s následným zvýšením jeho hodnoty.

Pokud je příznak přijatého příkazu aktivní, hlavní smyčka ještě nestihla zpracovat příkaz. V případě nového přijatého znaku přerušení znak odmítne a nastaví příznak přetečení.

```
void intc_USB_packet_received(void)
```

Ve vývojové verzi programu mikrokontroléru tato funkce neobsluhuje žádné přerušení. Komunikace po USB není realizována.

```
void intc_UDP_packet_received(void)
```

Ve vývojové verzi programu mikrokontroléru tato funkce neobsluhuje žádné přerušení. Komunikace po USB není realizována.

```
void intc_low_voltage_action(void)
```

Funkce obsluhující přerušení vyvolané detektorem snížení napětí. Ve vývojové verzi programu mikrokontroléru ale toto přerušení není nikdy vyvoláno.

```
void intc_RTC_timer_triggered(void)
```

Funkce obsluhující přerušení po GPIO vyvolané RTC modulem 3.2.5. Přerušení slouží ke zvětšení dvou proměnných

1. Čítač sekund od poslední synchronizace, který je používán při výpočtech současných nebeských souřadnic jednotky.
2. Čítač sekund od posledního přijatého příkazu, který je používán při automatickém ukládání konfigurace. Tento čítač je nulován při každém přijetí příkazu.

Pokud je hodnota čítače sekund od posledního příkazu rovna definované hodnotě, přerušení nastaví příznak uložení.

C.8 Komunikace s RTC

Funkce zprostředkovávající veškerou komunikaci s RTC modulem 3.2.5. MCU komunikuje s RTC modulem přes velmi rozšířenou sběrnici I2C. Ta je podporována přímo v hardware MCU za pomoci ASF knihoven, kterých je kódu využíváno. Zdrojový kód je uložen ve složce `./src/components/rtc` v následujících souborech

- `rtc_commands.c`
- `rtc_commands.h`
- `rtc_wrapper.c`
- `rtc_wrapper.h`
- `rtc.h`

Princip komunikace i organizace je téměř identický s komunikací s FPGAC.6.

C.8.1 Definice konstant a struktur

Definice konstant a datových struktur potřebných po komunikaci s RTC modulem je vyčleněna mimo soubory s funkcemi do samostatného hlavičkového souboru `rtc.h`. Tento soubor obsahuje návratové hodnoty a strukturu potřebnou pro komunikaci. Specifikem struktury pro komunikaci je zarovnání na 1 byte pro přenos po I2C.

C.8.2 Komunikace na nízké úrovni

```
int RTC_read(unsigned char addr, char * payload, unsigned char size)
int RTC_write(unsigned char addr, char * payload, unsigned char size)
```

Dvojice funkcí pro komunikaci s RTC modulem. Funkčnost je založena na ASF a využívá funkcí pro čtení a zápis po I2C. ASF také definuje strukturu zprávy, kterou tyto funkce přijímají. Komunikace je založena na správném zapsání adresy zařízení, ukazatele na místo v paměti, které má být odesláno, či kam má být načten výsledek a velikost tohoto paměťového bloku v bytech do struktury a zavolání odpovídající funkce zápisu nebo čtení s parametrem této struktury.

C.8.3 Komunikace z pohledu volajících funkcí

```
int RTC_read_time(double_t * julian_date)
int RTC_write_time(double_t julian_date)
```

Dvojice funkcí pro čtení a zápis času z RTC modulu. Funkce pracují s parametrem typu číslo v plovoucí řádové čárce. Tento parametr převádí pomocí funkce na časovou strukturu při zápisu, či získávají juliánské datum ze struktury při čtení. Funkce zapisují a čtou strukturu z bázové adresy v paměťovém prostoru RTC modulu. Tato adresa zůstává skryta volající funkci.

C.9 Soubor common.h

Detail popis souboru `common.h` je uveden v kapitole o mikrokontroléru 5.1.