TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKA TEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Karl Tarbe

Alglaadur ESTCube-1 käsu- ja andmehaldussüsteemile ja kaameramoodulile

Bakalaureusetöö (6 EAP)

Juhendaja: Meelis Roos, MSc

Autor: ..... "..."mai 2013
Juhendaja: ..... "..."mai 2013

Lubatud kaitsmisele
Professor: ..... "..."mai 2013

Tartu 2013

Sisukord

Lühendid

CAM *Camera module*

CDHS *Command and Data Handling System*

FRAM *Ferroelectric Random Access Memory*

I^{2}C *Inter-Integrated Circuit*

SPI *Serial Peripheral Interface*

SRAM *Static Random Access Memory*

1 Sissejuhatus

Aasta 2008 suvel alustati tudengisatelliidi projektiga. Esialgu pidi tegemist olema väga lihtsa satelliidiga, kus peal on võib-olla üks või kaks mikrokontrollerit, kuid asjad nii ei läinud. Inseneridele meeldib ikka üle töötada ja asju paremaks teha. Seetõttu sai ka Eesti esimesest satelliidist päris keeruline süsteem, mis koosneb mitmetest moodulitest. Kuna tegemist on üpris keerulise süsteemiga, peab ka selle juhtimiseks kasutatav tarkvara keeruline olema ja iga keerulise tarkvara puhul esineb tavaliselt vigu. Siinkohal muutubki aktuaalseks alglaadur, mille abil saab tarkvara uuendada.

Töö eesmärgiks on disainida ja realiseerida alglaadur, mille kaasabil on võimalik uuendada ESTCube-1 käsu ja andmehaldussüsteemil ja kaameramoodulil olevaid tarkvarasid. Täpsemalt peab alglaadur töötama kahel erineval moodulil, kus alglaadurit jooksutavad sarnased mikrokontrollerid.

Alglaadur ei pea suhtlema maajaamaga uue tarkvara vastuvõtmiseks. On tehtud eeldus, et olemasolev tarkvara võtab järgmise tarkvara versiooni vastu ja salvestab selle mikrokontrollerist väljajäävasse mällu ning annab alglaadurile käsu see mikrokontrolleri mällu kopeerida. Seega alglaaduri peamiseks ülesandeks on lugeda välisest mälust ja kirjutada mikrokontrolleri sisemisse mällu ning seejärel hüpata ülejäänud tarkvara käivitamise protseduuri. Väga oluline on ka see, et iga tegevuse kohta peetaks logi, mida saab hiljem vigade väljaselgitamiseks lugeda.

Üheks töö osaks on luua teek, mille abil saab alglaadurile käske ette valmistada ning alglaaduri poolt kirjutatud logi lugeda. Alglaaduri ülejäänud süsteemi integreerimine ei kuulu töö skoopi.

Töö autori panus on alglaaduri ja seda abistava teegi

- nõuete väljaselgitamine,
- tehniliste lahenduste disain,
- realiseerimine.

1.1 ESTCube-1

Tudengisatelliidi projekti käigus valminud satelliidi nimeks on ESTCube-1. Tegemist on kuupsatelliidiga, mille mõõtmed on 10×10×11.35 cm ja kaal on 1.048 kg. Satelliit vastab kuupsatelliidi standardile [1].

1.1.1 Missioon

Aastal 2006 leiutati uudne viis, kuidas kasutada päikeselt tulevat laetud osakeste voogu ehk päikesetuult liikumiseks. Selleks kasutatakse peenikesi

juhtmeid, mis pingestatakse päikesetuule suhtes. Sama laenguga osakesed tõukuvad ja kokkuvõttes hakkab päikesetuul neid juhtmeid koos lennumasinaga edasi tõukama. Analoogia põhjal tavalise tuulega kutsutakse seda juhtmete kimpu elektriliseks päikesepurjeks. Uudne lähenemine on umbes 100 korda efektiivsem kui hetkel planeetidevahelisteks lendudeks sobivaimad ioonmootorid [2].

ESTCube-1 missiooniks on esimest korda ajaloos katsetada elektrilist päikesepurje kosmoses. Satelliidi pardal olev juhtmete kimp kaalub 0,2 grammi ja on 20 meetri pikkune. Stardi ajal on juhtmed keritud kokku rulli peale, et stardimõõtmetesse ära mahtuda. Maa orbiidil pannakse satelliit pöörlema ja elektriline mootor kerib rulli lahti ning tsentrifugaaljõu tõttu kaugeneb juhtmete kimp satelliidist ühtlaselt.

1.2 Käsu- ja andmehaldussüsteem — CDHS

Käsu- ja andmehaldussüsteem ehk *Command and Data Handling System* (CDHS) on kavandatud satelliidi peamiseks pardaarvutiks. Peamised arvutused, mis määravad satelliidi positsiooni, toimuvad just CDHSis. Lisaks salvestatakse CDHSi igasuguseid erinevaid andmeid, mida sõltuvalt Maa pealt tulevatele käskudele ka tagastatakse.

Süsteem koosneb nii tarkvarast kui ka elektroonikast, mille peal tarkvara jooksutatakse. Antud bakalauresetöö raames pole põhitarkvara detailid eriti olulised, aga elektroonika on natuke oluline. Nimelt kuuluvad CDHSi juurde ka mitmed mälud, kus hoitakse erinevaid andmeid. Nendest mäludest oluliseks on ferroelektrilised juhupöördusega mälud (*Ferroelectric Random Access Memory* (FRAM)), sest need on radiatsioonirikkas keskkonnas mitu korda töökindlamad kui tavapärased välkmälud (*Flash memory*). Täpsemalt on mäludest juttu alapeatükis 1.4.

Protsessorituumaks on Cortex-M3, mis paikneb STMicroelectronics-i valmistatud mikrokontrolleris STM32F103, millest tuleb lähemalt juttu alapeatükis 1.5. Üheks märkimisväärseks omaduseks CDHSi puhul on see, et süsteemis on osaline riistvaraline dubleeritus: trükkplaadil on kaks identset mikrokontrollerit [3]. Esialgu jookseb neist üks, kuid kui tuvastatakse, et see mikrokontroller enam ei tööta, lülitatakse ümber teisele mikrokontrollerile. Vea tuvastamise ja ümberlülitamise eest vastutab toitesüsteem. Lülitatakse ümber ka andmesiinid, et mikrokontrollerist väljajäävad seadmed oleksid korraga ühenduses ainult ühe mikrokontrolleriga.

Toitesüsteemis pidi olema ka niiöelda väline valvekoer. Kui CDHSi mikrokontroller aegajalt ühte viiku ei vilguta, siis toitesüsteem järeldab, et CDHS on määramata olekus. Toitesüsteem taaskäivitab CDHSi sellelt ajutiselt toite äravõtmise abil. Seda signaali, mida CDHS genereerima peab, nimetatakse

südamelöögi signaaliks, sest kui seda signaali pole, siis on süsteem ,,surnud", nagu ilma südamelöökideta oleks surnud ka inimene.

1.3 Kaameramoodul — CAM

Kaameramoodul ehk *Camera module* (CAM) on disainitud tegema pilte kontrollimaks, kas päikesepurje väljakerimine õnnestub nii nagu soovitud. Sekundaarne ülesanne on teha Maast ja ka Eestist pilte, mida saaks kasutada teaduse populariseerimise eesmärgil. Pilte saab teha VGA lahutusega ehk 640×480 pikslit, mida on tänapäeval küll väga vähe, kuid piisavalt, et etteantud ülesandeid täita [4].

Sarnaselt CDHSiga juhib mooduli tööd mikrokontroller, mille tootjaks on STMicroelectronics. Mikrokontrolleriks on STM32F217, mis ei erine eriti CDHSi peal kasutatavast STM32F103st, kuid nendest erinevustest lähemalt saab lugeda alapeatükist 1.5.

Teine sarnasus CDHSiga on see, et ka CAMil on mikrokontrollerist väljajäävad mälud. Täpsemalt on alglaaduri jaoks oluline, et CAMil asub FRAM tüüpi mälu, millega mikrokontroller saab suhelda üle *Inter-Integrated Circuit* (I^2C) andmesiini. Ka CAMi jaoks on toitesüsteemis eraldi signaal, kuhu CAM peab südamelööke genereerima täpselt samamoodi, nagu seda teeb CDHS.

1.4 Mälud

Alglaadur puutub kokku kolme tüüpi mäludega, mida on kirjeldatud järgnevates jaotistes.

1.4.1 Juhupöördusega staatiline mälu — SRAM

Static Random Access Memory (SRAM) on hävimälu ehk seal olevad andmed kaovad, kui mälul ei ole voolu. Mälu asub mikrokontrolleri sees ja seda kasutab programm tavapärase operatiivmäluna. Kasutusel olevad mikrokontrollerid lubavad sealt ka koodi käivitada, aga seda funktsionaalsust alglaaduris ei kasutata.

1.4.2 Välkmälu — Flash

Tegemist on enimlevinud tehnoloogiaga säilmälude hulgas. Antud juhul paikneb seda tüüpi mälu mikrokontrolleri sees. Seal asub tarkvara, mida mikroprotsessor asub käivitama. Antud mikrokontrollerite puhul on mälu juhupöördusega nii kirjutamiseks kui ka lugemiseks, aga kirjutamiseks peab

mälu olema eelnevalt kustutatud. Kustutatud mälus iga biti väärtus 1. CDHSi peal kasutatava STM32F103 mikrokontrolleril toimub välkmälusse kirjutamine alati 16 biti kaupa. CAMil kasutataval STM32F217l saab valida järgnevate pikkuste vahel: 64, 32, 16 või 8 bitti [5]. See valik sõltub ka olemasolevast toitepingest, kuid et STM32F103 peal valikut ei ole ja alglaadur peab mõlema peal töötama, siis sarnasuse huvides on valitud STM32F217 korral samuti 16-bitine pikkus.

Välkmälu on jagatud mälulehtedeks, mis võivad olla kõik sama suurusega nagu STM32F103l või ka erineva suurusega nagu STM32F217l. Välkmälust kustutamine toimub antud mikrokontrollerite puhul kas terve mälu või ühe mälulehe kaupa. STM32F103 puhul nimetatakse mälulehti mälulehtedeks (*memory page*), aga STM32F217 dokumentatsioonis on kasutusel termin mälusektor (*memory sector*) [5, 6].

1.4.3 Ferroelektriline juhupöördusega mälu — FRAM

FRAM on sarnaselt välkmäluga säilmälu, kuhu jäävad andmed alles ka siis, kui seadmelt toide ära võtta. Nii CDHSil kui ka CAMil jääb FRAM mikrokontrollerist väljapoole ja sellega tuleb suhelda üle *Serial Peripheral Interface* (SPI) või I^2C andmesiini.

FRAMil on mitmeid eeliseid välkmälu ees:

- lubatud ülekirjutamise tsükleid on rohkem,
- voolutarve on kordades väiksem,
- kiirus on kordades parem,
- mälu on juhupöördusega,
- mälu on radiatsioonikindlam.

FRAMil on ka kaks miinust: esiteks on andmetihedus suhteliselt väike ja teiseks on selle hind kõrgem. Seda tüüpi mälu kasutatakse satelliidil, sest see on radiatsioonikindlam [7].

1.5 Mikrokontrollerid STM32F103 ja STM32F217

Mikrokontrollerid STM32F103 ja STM32F217 on STMicroelectronics-i tooted, mida kasutatakse vastavalt CDHSil ja CAMil. Kogu järgnev tekst siin jaotises viitab mikrokontrollerite tehnilistele manuaalidele [8, 9]. Esimene osa kirjeldab kontrollerite ühisosa ja teine kirjeldab nende erinevusi. Alglaaduri tööks mittevajalikud osad jäävad kirjeldamata.

Lisatäpsustena peab mainima, et käesolevas töös kasutatakse nimetusi STM32F103 ja STM32F217 konkreetsete mikrokontrollerite nimedena, kuid tegelikult tähistavad need kahte erinevat mikrokontrollerite seeriat. Samasse seeriasse kuuluvad mikrokontrollerid erinevad mälumahtude, sisend-/väljundviikude arvu või kontrolleri füüsilise pakendi poolest.

1.5.1 Sarnasused

Mõlemas kontrolleris on protsessorituumaks ARM arhitektuuriga Cortex-M3, mis on väga tihedalt seotud katkestustekontrolleriga. Täpsemalt saab selle kohta lugeda manuaalist [10].

Mikrokontroller erineb mikroprotsessorist selle poolest, et lisaks mikroprotsessorile on mikrokontrolleris olemas ka ülejäänud vajalik, et tööd teha: operatiivmälu, mälu tarkvara jaoks ja mitmed moodulid välise maailmaga ehk väliste seadmetega suhtlemiseks. Mõned moodulid, mida alglaadur oma töös kasutab, on õnneks kasutamise seisukohalt mõlemas kontrolleris täpselt samasugused.

Üheks selliseks mooduliks on kontrollsumma arvutamise üksus (*CRC calculation unit*), mille abil saab arvutada 32-bitist kontrollsummat(CRC-32). On mitmeid erinevaid standardeid, mille järgi CRC-32te arvutada. Antud moodul kasutab Ethernetis ja mujal kasutusel olevat polünoomi, mille väärtus kuueteistkümnendsüsteemis on 0x4C11DB7.

Kuna alglaadur peab suhtlema väliste mäludega üle I^2C ja SPI andmesiinide, on oluline ka see, et mõlema jaoks on eraldi moodulid olemas. Ilma eraldiasuva moodulita peaks raiskama protsessoriaega, et ükshaaval erinevaid sisend-/väljundviike juhtida, mis on küll võimalik, kuid ebaefektiivne, sest protsessor peab ühe andmevahetuse kella takti jooksul täitma mitmeid käske ja seega on andmevahetuse üldine kellasagedus kümneid kuni sadu kordi väiksem kui protsessori kellasagedus.

Käesoleva töö kontekstis on oluline, et alglaadimise protseduur on mõlemal kontrolleril samasugune ja see on kirjeldatud peatükis 3.

1.5.2 Erinevused

Peamiseks erinevuseks on see, et maksimaalne lubatud töösagedus protsessorile on STM32F103 korral 72MHz ning STM32F217 korral 120MHz. Alglaaduri seisukohalt pole see aga oluline, sest alglaadur kasutab kogu oma töö vältel madala sagedusega sisemist kellasignaali. Sisemise kellasignaali sagedus on 8 MHz.

Alglaaduri seisukohalt oluliseks erinevuseks on sisemise välkmälu erinev ülesehitus. STM32F103l on välkmälu jagatud kaheks mälupangaks (*memory*

bank), millele on juurdepääsuks erinevad registrid, aga STM32F217l on üks mälupank. Lisaks sellele on esimese kontrolleri puhul kõik mälulehed sama suurusega, kuid teises kontrolleris on mälulehtede suurused varieeruvad [5,6].

Erinev on ka sisend-/väljundviikude seadistamine. Erinev on nii lihtsalt viikude seadistamine kui ka mingite viikude määramine riistvara moodulitele kasutamiseks.

2 Süsteemi nõuded

Käesolevas peatükis on välja toodud alglaadurile alguses paika pandud nõuded ja kasutusjuhud. Need sündisid koostöös CDHSi arendaja Indrek Sünteriga. Lisasoovitusi andsid akadeemilised nõustajad Viljo Allik ja Tõnis Eenmäe. Siin peatükis pole eriti räägitud CAMist, sest alglaadur on mõeldud eelkõige CDHSile. CAMil kasutatakse lihtsalt sama alglaadurit.

2.1 Alglaaduri käsud

2.1.1 Käsk kindla tarkvara alglaadimiseks

Käsk on vajalik, et vahetada kasutatavat tarkvara. Käsu abil saab valida, millises pesas olev tarkvara alglaaditakse.

2.1.2 Käsk tarkvara kopeerimiseks

Käsk on vajalik, et saaks teostada tarkvara uuendamist, kui CDHS töötab. Käsu eesmärk on kopeerida tarkvara tõmmis välisest mäluseadmest sisemisse välkmällu. Enne kopeerimist tuleb valideerida tarkvara tõmmise terviklikkus.

2.2 Alglaaduri käskude loend

Kuna alglaaduriga ei suhtle vahetult teine seade, vaid põhitarkvara ise, siis on vajalik, et oleks olemas loend, kuhu põhitarkvara paneb alglaadurile täitmiseks mõeldud käsud.

2.2.1 Käskude lisamine loendisse

Käske peab saama loendisse lisada, sest muidu ei oleks võimalik alglaadurile käske anda. Käskude lisamise eelduseks on see, et põhitarkvara töötab.

2.2.2 Vigade logimine

Vigade logimine on vajalik, et vigadest teada saada. Kui mõne alglaaduri käsu täitmine ebaõnnestub, tuleb vea info logisse salvestada.

2.2.3 Vaikimisi alglaadimine

Enamasti on alglaaduri käskude loend tühi ja selleks ajaks on vaja head ja ohutut vaikimisi käitumist, mis alglaadiks olemasoleva tarkvara.

2.2.4 Õnnestumise logimine

Kui midagi alglaaduri töös halvasti läheb, siis on vaja teada, mille alglaadur edukalt juba korda saatis.

2.2.5 Käskude loendi valideerimine

Radiatsiooni või mingi muu nähtuse tõttu võib käskude loend kahjustada saada. Vigaste käskude täitmise vältimiseks tuleb käskude loendi terviklikkus üle kontrollida.

2.3 Alglaaduri töö

2.3.1 Välisest mälust lugemine

Sisemises välkmälus pole piisavalt ruumi, et hoida piisavat arvu tagavara tarkvara tõmmiseid. Seega peab alglaadur suutma väliste mäluseadmetega suhelda.

2.3.2 Sisemisse välkmällu kirjutamine

Mikrokontroller suudab koodi jooksutada ainult sisemisest välkmälust või SRAMist, seega peab alglaadur suutma paigutada tarkvara sinna, kus seda saab käivitada ja hoiustada.

2.3.3 Tarkvara kontrollsumma kontrollimine kopeerimisel

Tarkvara võib välises mälus kahjustada saada, seega tuleb enne kopeerimist kontrollida, kas kontrollsumma klapib, et tagada tarkvara tõmmise terviklikkus.

2.3.4 Tarkvara kontrollsumma kontrollimine alglaadimisel

Tarkvara võib sisemises välkmälus kahjustada saada ja vigast tarkvara ei tohiks käima panna.

2.3.5 Südamelöögi pakkumine

Alglaadur peab genereerima südamelöögi signaali, et toitesüsteem ei püüaks toitepinget eemaldades CDHSile taaskäivitust teha.

2.3.6 Tarkvarapesa valimine kasutades CDHS_FIRM signaali

Vea korral peab olema toitesüsteemil võimalus valida teine CDHSi tarkvara tõmmis. Selleks muudab toitesüsteem signaali CDHS_FIRM väärtust. Alglaadur saab selle väärtuse teada vastavat sisendviiku lugedes.

2.3.7 Kahe mikrokontrolleri toetamine

Alglaadur peab töötama nii STM32F103l, mis on CDHSil, kui STM32F217l, mis on CAMil.

2.4 Kasutuslood

Siin on esitatud esialgsed kasutuslood, mis kirjeldavad seda, kuidas ,,tellija" ehk CDHSi peaarendaja, Indrek Sünter alglaaduri kasutamist ette nägi.

2.4.1 Tavapärane alglaadimine

1. Protsessor laadib alglaaduri.
   (a) Kõik lisaseadmed ja riistvaramoodulid lülitatakse välja.
   (b) Kõik sisend-/väljundviigud seatakse analoogsisendiks.
   (c) Valvekoera taimer (*Watchdog Timer*) seadistatakse.
   (d) Lihtsad I^2C FRAMi ajurid laaditakse.
   (e) Kontrollsumma moodul seadistatakse.
   (f) Südamelöögi signaal seadistatakse.
   (g) Alglaadur kontrollib iseenda kontrollsummat.
   (h) Alglaadur võtab esimese käsu, valideerib ja täidab selle.
   (i) CDHSi tarkvara alglaaditakse.
   (j) CDHSi tarkvara loeb alglaaduri logi ja kopeerib sealt huvitavad elemendid oma vigade logisse.

2.4.2 CDHSi tarkvara uuendamine

1. Uue tarkvara tõmmis hoiustatakse välises FRAMis ning CDHS taaskäivitatakse koos
   (a) käsuga alglaadurile, et kopeerida see mikrokontrolleri sisemisse välkmälusse.
2. Alglaadur võtab esimese käsu ja täidab selle.
   (a) Käsuks on kopeerida tarkvara tõmmis välimisest FRAMi pesast X sisemisse välkmälu pessa Y.
      i. Arvutatakse pesas X oleva tõmmise kontrollsumma ja võrreldakse seda pesa X päises olevaga.
         A. Kopeeritakse tarkvara tõmmis pesast X pesasse Y.
         B. Kontrollitakse tarkvara tõmmist pesas Y ja vajadusel korratakse kopeerimist kuni kolm korda.
         C. Logitakse õnnestumine.

2.4.3 Välisest FRAMist lugemine ebaõnnestub

1. Alglaadur käivitatakse koos käsuga kopeerida tarkvara tõmmis välisest FRAMist.
2. Alglaadur üritab lugeda välisest FRAMist.
   (a) Oletame, et FRAM ei vasta, või I^2C andmesiin on hõivatud.
3. I^2C viga logitakse.
4. Käskude loend tühjendatakse.
5. Alglaadur alglaadib vaikimisi pesas oleva tarkvara.

2.4.4 Kontrollsumma erinevus tarkvara tõmmisel

1. Alglaaduril käsitakse opereerida tarkvara tõmmisel (kas kopeerida või alglaadida).
2. Alglaadur arvutab tarkvara tõmmise kontrollsumma.
3. Alglaadur võrdleb seda tarkvara tõmmise päises oleva kontrollsummaga.
   (a) Oletame, et need ei ühti.

4. Viga logitakse.
5. Käskude loend tühjendatakse.
6. Alglaadur valib vaikimisi tarkvara tõmmise ja proovib seda alglaadida.

2.4.5 Käskude loendis on tundmatu käsk

1. Alglaadur vaatab käskude loendit.
    (a) See ei ole tühi.
2. Alglaadur võtab esimese käsu loendist.
3. Alglaadur proovib seda verifitseerida, kuid ebaõnnestub.
4. Alglaadur logib vea.
5. Käskude loend tühjendatakse.
6. Alglaadur valib vaikimisi tarkvara tõmmise ja proovib seda alglaadida.

2.4.6 Soovitud tarkvara tõmmis valitakse FIRMW viigu abil

1. Alglaadur vaatab käskude loendit.
2. Käskude loend on tühi.
3. Alglaadur vaatab FIRMW viiku.
    (a) See on madal?
        i. Laaditakse vaikimisi pesas olev tarkvara.
    (b) See on kõrge?
        i. Laaditakse tagavarapesas olev tarkvara.

2.4.7 Alglaaduril ebaõnnestub vea logimine

1. Midagi juhtub ja alglaaduril ebaõnnestub vea logimine.
    (a) Vigade logi on täis?
        i. Alglaadur ei kirjuta vanemaid sissekandeid üle.
2. Alglaadur jätkab tööd probleemi ignoreerides.

2.4.8 Spetsiifilise tarkvara tõmmise alglaadimine

1. CDHS käivitab alglaaduri käsuga laadida tarkvara kindlast pesast.
2. Alglaadur verifitseerib käsu.
3. Alglaadur logib õnnestumise.
4. Alglaadur tühjendab käskude loendi.
5. Alglaadur alglaadib soovitud pesas oleva tarkvara tõmmise.

2.4.9 Alglaaduri kontrollsumma erinevus

1. Alglaadur arvutab enda kontrollsumma.
2. Alglaadur võrdleb seda varem salvestatuga.
   (a) Oletame, et need ei ole võrdsed.
3. Viga logitakse.
4. Alglaadur valib vaikimisi tarkvara tõmmise ja proovib seda alglaadida.

2.5 Muutused töö käigus

Nagu peaaegu igale tarkvaraprojektile kohane, muutusid ka selle projekti käigus nõuded. Nõuded täpsustusid peamiselt seetõttu, et esialgsete nõuete koostamisel ei oldud teadlikult väga põhjalik ja esialgsed nõuded jätsid päris mitmed detailid lahtiseks. Samuti on vasturääkivusi esitatud kasutuslugudes.

2.5.1 Töö käigus tekkinud täpsustused

Käskude loendi hoidmiseks valiti väline FRAM, sest see on radiatsioonikindlam kui välkmälu. Kui peaks juhtuma, et välise FRAMiga ei saa suhelda, siis pole alglaaduril võimalik sealt tarkvara kopeerida. Tarkvaratõmmise valimiseks saab kasutada ka toitesüsteemi, mis muudab FIRMW viigule mineva signaali väärtust.

Samuti otsutati, et ka kopeerimine toimub just FRAMist sisemisse välkmälusse. Väline välkmälu oli alternatiivse koht tarkvara tõmmiste hoiustamiseks. FRAMi kasuks otsustati, sest siis ei pea realiseerima suhtlust välise välkmäluga ja alglaaduri keerukus ei kasva selle võrra.

Algselt leiti ka veel seda, et nii CAMil kui ka CDHSil on I^2C siinil olev FRAM. See tähendas, et oli vaja realiseerida ainult I^2C suhtlus. Hiljem

avastati, et CDHSi I^2C andmesiinil olev FRAM on liiga väike, et seal tarkvara tõmmiseid hoida. Seetõttu tuli realiseerida ka SPI siinil suhtlus FRAMiga, sest CDHSi peal oli SPI siinil piisava suurusega FRAM. Teisest küljest CAMil ei ole SPI siinil asuvat FRAMi ja seega jäädi seal I^2C siinil oleva FRAMi juurde.

2.5.2 Kasutuslugude muutused

Kasutusloos ,,2.4.1 Tavapärane alglaadimine" pole vaja teha kahte esimest tegevust, sest pärast mikrokontrolleri resetti on niigi kõik lisaseadmed välja lülitatud ja enamik sisend-/väljundviikudest on seadistatud analoogsisendiks [8, 9]. Lisaks leiti töö käigus, et sisemist valvekoera taimerit ei peaks kasutatama, sest kui CDHSi tarkvara kokku jookseb, teeb valvekoer mikrokontrollerile taaskäivituse. Kui tarkvara järjepidevalt uuesti hangub, siis jääb süsteem taaskäivituste tsüklisse. Kui nüüd selle tsükli käigus ka südamelöögi signaali genereeritakse, siis ei saaks toitesüsteem probleemist teada ja ei oskaks ümber lülitada teisele tarkvara pesale või CDHSi puhul teisele mikrokontrollerile. Samuti on vajab kohendamist ka punkt (d), sest CDHSil saab FRAMile ligi I^2C asemel üle SPI andmesiini.

Kasutusloos ,,2.4.2 CDHSi tarkvara uuendamine" ei ole vaja välimises FRAMis kindlat pesa määrata, vaid selle asemel kasutatakse lihtsalt mäluaadressi, millelt tarkvara tõmmis leitakse.

Kasutuslugu ,,2.4.3 Välisest FRAMist lugemine ebaõnnestub" tuleb ka muuta, sest FRAM ei pruugi alati olla I^2C siinil ning enne kopeerimise käsku tuleb FRAMist lugeda käskude loend. Samuti on käskude loendi kustutamiseks vaja FRAMi kirjutada.

Kasutusloost ,,2.4.6 Soovitud tarkvara tõmmis valitakse FIRMW viigu abil" eemaldati tingimus, et käskude loeng peab olema tühi. Lihtsalt iga kord, kui on juttu vaikimisi pesas oleva tarkvara alglaadimisest, siis valitakse pesa vastavalt selle viigu väärtusele. Koodis on tagavarapesa asemel leebem nimetus: sekundaarne pesa.

3 Alglaadimine

3.1 Tavapärane programmi käivitamine

Käesolevas alapeatükis kirjeldatakse protsessori tööd programmi käivitamisel.

Mikrokontroller üritab ennast tööle panna, kui toide olemas on. Esmalt lülitatakse sisse sisemine 8-MHz-ne ostsillaator, mille signaali neljandal tõusval frondil loetakse konfigureerimise viikude väärtused. Nende väärtuste põhjal otsustab mikrokontrolleri sisemine juhtloogika, millisest kohast programm

käivitada. Võimalikud variandid on: põhi-välkmälu, süsteemi mälu ja staatiline juhupöördlusega mälu (SRAM). Enim kasutatakse kahte esimest: põhi-välkmälu seal oleva programmi käivitamiseks ja süsteemi mälu. Süsteemi mälu on kaitstud osa põhi-välkmälu lõpust, kuhu tehases paigaldatakse alglaadur, mille abil saab programmeerida ülejäänud välkmälu. Tehases paigaldatud alglaadur võimaldab mikrokontrollerile oma tarkvara peale laadida.

Edasist protsessi vaatleme tehes oletuse, et programm on juba kontrolleri välkmälus olemas ja alglaadimist alustatakse põhi-välkmälust. Välkmälu aadress on tegelikult 0x800 0000, aga viikude oleku tõttu peegeldatakse välkmälu aadressiruum ka aadressile 0x0000 0000. See tähendab, et mäluoperatsioon aadressiga 0x0000 0000 annab sama tulemuse, mis ta annaks aadressiga 0x800 0000. Protsessor võtab aadressil 0x0000 0000 oleva väärtuse ja kasutab seda pinu viida (*stack pointer*) algväärtustamiseks. Seejärel alustatakse käskude täitmist lähtestamise rutiinist, mille algusaadressi võtab protsessor aadressilt 0x0000 0004 [8].

Lähtestamise vektoril on tavaliselt assembleris kirjutatud protseduur, mis kopeerib programmi andmete sektsiooni välkmälust SRAMi, täidab ülejäänud programmi jaoks vajaliku ala SRAMist nullidega, kutsub välja C koodis kirjutatud protseduuri, mis initsialiseerib süsteemi kella, ja siis annab juhtimise üle C koodis kirjutatud *main* protseduurile. Satelliidi peal ei kasutata üldist kella seadmise koodi ja seega see samm jäetakse vahele ning kella seadmine kutsutakse välja *main*ist.

See, mida tarkvara edasi initsialiseerib, on juba tarkvara spetsiifiline. Tõenäoliselt initsialiseeritakse mõned riistvaralised moodulid näiteks USART või SPI ning hakatakse siis rakendusele spetsiifilist kontrolltsüklit täitma. Satelliidil ESTCube-1 kasutatakse nii CAMil kui ka CDHSil põhitarkvara osana reaalaja operatsioonisüsteemi FreeRTOS.

3.2 Programmi alglaadimine koos alglaaduriga

Alglaadur ise käitub alglaadimise seisukohalt nagu iga teinegi programm ja laaditakse täpselt samamoodi nagu eelmises alapeatükis kirjeldatud. See tähendab seda, et alglaaduri katkestusvektorite tabel peab asuma välkmälu alguses, sest sealt hakkab protsessor seda otsima. Kuna üks osa alglaadurist asub juba välkmälu alguses, on ka teised osad sinna järele paigutatud, mis omakorda tähendab, et tegelik põhiprogramm peab nüüd asuma kusagil mujal kui välkmälu alguses ja sellega peab arvestama põhiprogrammi linkimisel.

Kui alglaadur on oma muud ülesanded juba täitnud, on selle viimaseks tööks põhitarkvara käivitamine. Selleks jäljendab alglaadur protsessori käitumist, kuid kõigepealt on vaja teada, mis aadressil põhitarkvara katkestusvektorite tabel asub. See aadress on eelnevalt kokku lepitud ja sissekodee-

ritud.

Enne põhitarkvara töölepanemist tuleks puhtaks teha pinu. Alglaaduri enda töö ajal sinna salvestatud väärtusi pole põhiprogrammil vaja, sest ideaalis ei tohiks põhiprogramm arugi saada, kas see käivitati otse või alglaaduri abiga. Pinu puhtaks tegemine tähendab lihtsalt pinu viida algväärtustamist, sest nii hakatakse edaspidi pinus juba olevaid andmeid üle kirjutama ja kui programm on korrektne, siis midagi sellist ei loeta, mida programm ise pole juba üle kirjutanud. Algväärtustamiseks kasutatakse põhitarkvara katkestusvektorite tabeli esimest elementi, mis näitab pinu ülemist aadressi.

Kuigi mikrokontrolleri käivitamise hetkel alustatakse tööd kindlast kohast, ei pea terve katkestusvektorite tabel välkmälu alguses olema. Katkestustega tegeleb eraldi riistvara moodul NVIC (*Nested Vectored Interrupt Controller*). Lisaks on sellega tihedalt seotud teine moodul SCB (*System Control Block*), kus asub seadistatav register VTOR (*Vector Table Offset Register*), mille väärtust kasutatakse katkestusvektorite tabeli aadressina [10]. Vaikimisi on seal väärtus 0x0000 0000, mida kasutatakse alglaaduri laadimiseks, kuid alglaadur seab selleks väärtuseks põhiprogrammi katkestusvektorite tabeli aadressi. Kui seda ei seadistataks, siis kutsutaks põhiprogrammi katkestuse ajal välja alglaaduri katkestusvektoreid, mis on tegelikult spetsifitseerimata, sest alglaadur ei kasuta katkestusi. Lisaks sellele on vaikimisi katkestuse teenindamise rutiin lihtsalt lõputu tsükkel, seega jääks kontroller katkestusse kinni ja halvatud ei oleks ainult põhitarkvara katkestuse teenindamine vaid kogu põhitarkvara töö peatuks.

Kui pinu viit ja katkestusvektorite tabeli asukoht on ära seadistatud, jääb üle lugeda katkestusvektorite tabelist põhiprogrammi lähtestamise vektor ehk lähtestamise rutiini algusaadress ja mikrokontrolleri töö sinna edasi juhtida. Selleks tehakse C-koodis funktsiooni viit, mis väärtustatakse lähtestamise vektoriga ning seejärel kutsutakse see välja. Siit edasi on juhtimine juba alglaaditava tarkvara käes ja alglaadur ei tee enne järgmist süsteemi taaskäivitust enam midagi.

Edasine toimub sarnane tegevus nagu tavapärase programmi käivitamise juures, kus lähtestamise vektoril olev protseduur kopeerib välkmälus oleva põhiprogrammi andmete sektsiooni SRAMi, tühjendab SRAMi initsialiseerimata andmete jaoks ja seejärel kutsub välja põhitarkvara *main* protseduuri.

3.3 Programmi ettevalmistamine

Selleks, et põhitarkvara saaks kasutada alglaaduriga, tuleb eelnevalt programm niimoodi linkida, et arvestatakse sellega, millisele mäluaadressile tarkvara hiljem pannakse. Kui programm linkida kokku aadressi 0x800 0000 jaoks ja alglaadur paigutab programmi aadressile 0x800 3000, siis on prog-

rammi katkestusvektorite tabelis valed väärtused. Näiteks taimeri katkestuse käsitlemise funktsioonile vastavaks aadressiks katkestuste tabelis võib-olla välkmälu alguses paiknev aadress 0x800 09F0. Taimerikatkestuse korral võtab protsessor katkestusvektorite tabelist väärtuse 0x800 09F0 ja hakkab sealt käske täitma, kuid tõenäoliselt paikneb seal mingi alglaaduri osa.

Programmi linkimise ajal peab teadma aadressi, kuhu alglaadur selle programmi paneb. Lisaks aadressi teadmisele tuleb ka linkijale öelda, et ta programmi just selle aadressi jaoks lingiks.

4 Muud realisatsioonidetailid

4.1 Riistvara moodulitele juurdepääs

STMicroelectronics annab koos oma mikrokontrolleritega kaasa ka abiteegid, kus on funktsioonid, mis lihtsustavad riistvaramoodulite kasutamist. Alglaaduris neid ei kasutata, sest need lisavad alglaaduri ja riistvara vahele ebavajaliku lisakihi, mis raskendab silumist. Miinuseks on ka see, et abiteegid tõstavad alglaaduri keerukust ning suurust.

4.2 Alglaaduri kontrollsumma kontrollimine

Alglaaduri enda kontrollsumma arvutamiseks kasutatakse riistvaramoodulit, millest oli juttu jaotises 1.5.1.

Kui kontrollsumma on arvutatud, tuleb seda millegagi võrrelda. Selleks on linkija skriptis alglaaduri lõppu lisatud üks 4 baidi piirile joondatud 32-bitine muutuja. Muutuja on 4 baidi piirile joondatud, sest kontrollsumma arvutamise moodul arvutab just 4 baidi ehk 32 biti kaupa. Selle muutuja algväärtustamisel on tehtud lisaeeldus, et alglaadur ei saa kannatada enne esimest töölepanemist. Seda eeldust on vaja, sest linkija paneb selle väärtuseks lihtsalt 0xFFFF FFFF ja esimesel korral, kui alglaadur oma kontrollsumma arvutab, programmeeritakse välkmälus see väärtus üle tegeliku kontrollsummaga. Kusjuures esialgne väärtus 0xFFFF FFFF ei ole niisama valitud. See väärtus on sellepärast, et selline on välkmälu kustutatud asendis, ja nii on võimalik see üle programmeerida ilma tervet sektorit eelnevalt kustutamata.

4.3 Tarkvara tõmmise päis

Tarkvara tõmmist võib vaadelda, kui lihtsalt ühte suurt kogu andmeid, mille alguses on katkestusvektorite tabel. Alglaaduri seisukohast oleks vajalik, et oleks teada ka nende andmete pikkus ja kontrollsumma. Selleks lisatakse

tarkvara tõmmise ette nii FRAMis kui ka välkmälus pisike päis, kus on kolm 32 bitist välja: suurus, versioon ja kontrollsumma. Versioonivälja alglaadur kunagi ei puutu, aga põhitarkvara võib sellest kasu saada.

Päise lisamine aga tekitas ühe lisaprobleemi. Nimelt on välkmälus valitud tarkvarapesad selliselt, et need algaksid välkmälu lehtede piirilt, et neid oleks mugav kustutada. Need on suhteliselt ümmargused aadressid, kus ümmargune tähendab seda, et aadressi madalamad bitid on nullid. Enne päise lisamist töötas põhitarkvara normaalselt, kuid pärast päise lisamist esinesid probleemid põhitarkvara töös. Põhjuseks oli katkestusvektorite tabeline nihkumine päise võrra edasi, aga katkestusvektorite tabel peab olema ka mõnevõrra ümmargusel aadressil: 9 madalamat bitti peavad olema nullid [10].

Kompenseerimiseks lisati päise ja ülejäänud tõmmise vahele 500 baiti polstrit, mis nihutas katkestusvektorite tabeli piisavalt ümmargusele aadressile. Selline lahendus küll töötab, kuid raiskab ära 500 baiti. Alternatiivina oleks saanud panna selle päise pärast katkestusvektorite tabelit, sest katkestusvektorite tabel on fikseeritud pikkusega, kuid see oleks rohkem tööd võtnud ja samuti oleks siis pidanud eraldi arvutama kontrollsumma katkestusvektorite tabelist ja ülejäänud osast, mis jääks teisele poole päist.

Tarkvara tõmmis koos päise ja polsterdusega on ühesugune nii FRAMis kui ka välkmälus, kuigi FRAMis pole seda polsterdust vaja. Tarkvara kontrollsumma arvutamisel ei arvutata sisse päist ega polsterdust. Polsterdust saab kasutada, et seal veel mingeid metaandmeid tarkvara tõmmise kohta hoida, sest ka see polsterdust kopeeritakse alglaaduri poolt kohusetundlikult FRAMist välkmällu.

4.4 Käskude loend

Käskude loend asub FRAMis. Kasutades alglaaduri abiteeki saab sinna kirjutada maksimaalselt 3 käsku ja valida on kahe käsutüübi vahel: kopeerimine või alglaadimine. Iga käsk koos oma parameetritega kirjutatakse FRAMi kahekordselt, kusjuures teine kord on kõik bitid inverteeritud.

Et vältida vanade käskude loendisse sattumist, kirjutatakse iga käsu lõppu veel kaks baiti, mille väärtuseks on 0xFF. See tagab selle, et käskude loendi algusesse uue käsu kirjutamisel kirjutavad need kaks lisabaiti üle käskude loendis varem olnud vana käsu alguse, mistõttu alglaadur ei loe kaugemale viimati kirjutatud käskudest ka siis, kui unustatakse käskude loend eelnevalt tühjendada.

Käskude loendi tühendamine tähendab lihtsalt maksimaalse suurusega alale 0xFFide kirjutamist. Seejuures ei eristata käskude loendit lugedes, kas käskude loend oli tühi või lihtsalt esimene bait loendis oli vale.

Ühe puudusena võib välja tuua selle, et käskude loendit lugedes ei kontrollita, kas on ainult kolm käsku. Seetõttu saab FRAMi kirjutada ka sellise käskude loendi, kus on näiteks sadu kopeerimiskäske, aga see ei ole probleem, sest käskude loendit ei loeta korraga mälusse vaid täidetakse käskhaaval.

4.5 Südamelöögi signaali genereerimine

Südemelöögi signaali saab genereerida kasutades selleks riistvaralist taimerit, kuid see oleks vale, sest tarkvara võib kokku joosta niimoodi, et taimer genereerib signaali edasi. Seega on õigem tarkvarast manuaalselt seda signaali genereerida. Pealegi signaal ise on lihtne. Lihtsalt ühe väljundviigu väärtust tuleb muuta piisavalt tihedalt. Ning pole oluline, kui seda tehakse ebavajalikult tihti.

Viigu väärtust muudetakse vahetult enne põhitarkvara alglaadimist ja kahes funktsioonis, mida kasutatakse, et oodata mingite operatsioonide lõppemist. Nendest esimene on välkmälu operatsioonide jaoks ja teine I^2C FRAMi jaoks. Siit aga tuleb välja üks probleem, mis on kirjas jaotises 7.2.

4.6 Alglaaduri logi

Alglaaduri logi salvestatakse ühte välkmälu lehte. Igal alglaaduri käivitamisel leitakse kahendotsinguga üles seal lehes koht, kuhu logi kirjutamine pooleli jäi ja jätkatakse sealt logi täitmist. Kui juhtub, et käivitamise hetkel on logi peaaegu täis, siis võetakse viimased 10 logi elementi, kopeeritakse need SRAMi, kustutatakse mäluleht välkmälus ära, ja siis kirjutatakse need 10 logi elementi logi algusesse. Samuti hoolitsetakse, et kui kuidagi peaks töö käigus logisse rohkem kirjutatama, kui logis ruumi on, siis jäetakse ikkagi logi alast välkmälus väljapoole kirjutamata. Mittemahtuvad logikirjed visatakse lihtsalt minema, sest normaalse töö käigus ei tohiks logi kunagi täis saada.

4.7 Kasutatavad tööriistad

Alglaaduri realiseerimiseks valiti C programmeerimiskeel, sest see sobib hästi manussüsteemide programmeerimiseks. Kasutatavale arhitektuurile on olemas kättesaadav C kompilaator. Tööriistavööna kasutati Sourcery G++ Lite'i, mis on *GNU Compiler Collection*i(GCC) kommertsiaalne versioon ARM arhitektuuri jaoks.

5 Komponendid

CDHSi ja CAMi tarkvara tegemiseks kasutatakse Eclipse'i. Alglaaduri tegemiseks kasutati ka Eclipse'i, et hoida sarnast ülesehitust. Töö on jagatud kaheks projektiks, millest üks on alglaadur ise ja teine on abiteek, mille abil saab alglaadurile käske anda, käske lugeda ja alglaaduri logi lugeda.

5.1 Alglaadur

Alglaaduri projekt koosneb mitmetest failidest, mille hulgas on 9 C-faili, 4 päisefaili, 2 assembleri faili ja 2 linkimise skripti. Kuna alglaadur peab töötama kahel erineval platvormil, siis on platvormispetsiifilised kohad koodis kaitstud eeltöötluse makrodega (*preproccesor macros*). Eeltöötluse jaoks defineeritakse sümbolid vastavalt Eclipse projektis olevale ehitamise seadistusele (*build configuration*). See tähendab, et kompileerides alglaadurit näiteks CDHSi jaoks jäävad kompilaatoril nägemata need osad koodist, mis on mõeldud ainult CAMile, sest eeltöötluse käigus võetakse need välja.

5.1.1 Assembleri failid — startup_stm32fxxxx.S

Failid **startup_stm32f10x_xl.S** ja **startup_stm32f2xx.S** on tegelikult peaaegu täpselt samasugused. Neid on kaks tükki, sest alglaadur peab töötama kahel erineval mikrokontrolleril. Neis failides on kirjas katkestusvektorite tabel, mis on kasutatavatel mikrokontrolleritel erinev. Lisaks sellele on neis failides põhiline andmete SRAMi kopeerimine, SRAMi nullide kirjutamine ja C-koodis oleva *main*i väljakutsumine, millest oli juttu alapeatükis 3.1.

5.1.2 Süsteemi päised — stm32fxxx.h

Päised **stm32f10x.h** ja **stm32f2xx.h** on mikrokontrolleritele STM32F103 ja STM32F217 vastavad abifailid, milles on defineeritud eeltöötluse makrod ja struktuurid, mis teevad riistvaramoodulite registrite kasutamise oluliselt lihtsamaks. Need failid on saadaval mikrokontrolleri tootja käest.

5.1.3 Cortex-M3 päis — core_cm3.h

Selles päises on defineeritud protsessorituumaga seotud registritele juurdepääsemist abistavad makrod ja struktuurid. Lisaks sellele on seal prototüübid funktsioonidele, mis annavad C-koodist juurdepääsu spetsiaalsetele masinkoodi käskudele, mida kompilaator omapead ei kasuta. Päisega käib koos realisatsioonifail **core_cm3.c**, kus on antud realisatsiooni nendele spetsiaalsetele käskudele kasutades selleks C-faili kirjutatavat assemblerit. Nii päis kui

ka realisatsioon on saadaval Cortex-M3 protsessorituuma disainijalt ARM Holdings-ilt.

5.1.4 Peamine päisfail — bootloader.h

Esimese asjana kaasab see päis endaga vastavalt eeltöötluse muutujatele õige süsteemi päise. Kuna peaaegu kõik teised C-failid kaasavad selle faili, saavad nad nii ka ligipääsu õigetele süsteemi päisetele, mille abil riistvaramoodulitega suhelda. Lisaks sellele on antud failis prototüübid kõikidele alglaaduris kasutatavatele funktsioonidele, mida kasutatakse väljaspool funktsiooni defineeriva faili skoopi. Tavalised tehakse küll nii, et ühe tarkvaramooduli kohta on üks päisfail ja üks realisatsioonifail, aga nii oleks tekkinud palju lühikesi päisfaile, mis teevad koodi lugemise asjatult raskemaks.

5.1.5 Põhifunktsionaalsus — bootloader.c

Selle failis on realiseeritud alglaaduri põhifunktsionaalsus. Seal on protseduurid kindlal aadressil oleva tarkvara alglaadimiseks, tarkvara kopeerimise käsu ja kindlas pesas oleva tarkvara alglaadimise käsu täitmiseks.

5.1.6 Sisemise välkmäluga opereerimine — flash.c

Failis on realiseeritud välkmälu kustutamine ja programmeerimine. Lugemist pole vaja, sest välkmälu asub mikrokontrolleri aadressiruumis ja sealt lugemine tähendab lihtsalt mingilt aadressilt lugemist. Realisatsioon abstraheerib ära mikrokontrollerite erinevused.

5.1.7 I^2C FRAMiga suhtlemine — fram.c

Fail realiseerib I^2C suhtluse välise FRAMiga. Antud fail töötas nii CDHSil kui ka CAMil, kuid hiljem selgus, et CDHSi peal peab siiski kasutama SPI siinil olevat FRAMi. Sellest ajast on ka faili nimi lihtsalt viide mälu tüübile. Hiljem lisandus kood ka SPI siinil oleva FRAM jaoks. Fail on kasutusel ainult CAMil.

5.1.8 SPI FRAMiga suhtlemine — fram_spi.c

Failis on realiseeritud samad funktsioonid FRAMiga suhtlemiseks, kuid seda üle SPI andmesiini. Fail on kasutusel ainult CDHSil.

5.1.9 Sisend/väljund viikude seadmine — gpio.c

Antud fail lülitab sisse lisaseadmed neile tööks vajaliku kellasignaali lubades, seadistab sisend-/väljundviigud ja määrab need vajadusel riistvaramoodulitele. Samuti on seal funktsioonid tarkvara valimise viigu lugemiseks ja südamelöögi tekitamiseks. Antud failis on vastavad funktsioonid mõlemale mikrokontrollerile, kuid need erinevused on ära abstraheeritud.

5.1.10 Alglaaduri logimine — logging.c

Fail realiseerib funktsionaalsuse, mis on kirjeldatud alapeatükis 4.6.

5.1.11 Alguspunkt — main.c

Siin failis asub *main* protseduur, mis initsialiseerib kõigepealt sisend-/väljundviigud, logi ja välise FRAMiga suhtlemiseks vajaliku SPI või I^2C mooduli. Seejärel kontrollib protseduur alglaaduri enda kontrollsummat. Pärast seda täidab *main* käskude loendis olevad käsud ja lõpetuseks teeb vaikimisi alglaadimise.

5.1.12 Testid — tests.c

Fail koosneb mõnedest funktsioonidest, mida kasutati testimisel. Alglaaduris neid tegelikult vaja ei ole. Siin on kood FRAMist kirjutamise ja lugemise testimiseks, logisse kirjutamiseks ning käskude loendisse testkäskude kirjutamiseks.

5.1.13 Linkija skriptid — stm32_fx_flash.ld

Tegemist on kahe failiga, mida vastavalt platvormile kasutatakse alglaaduri linkimiseks. Failides on defineeritud mäluregioonid, mis vastavad alglaaduri logile ja põhitarkvara pesadele, koos muutujatega, et alglaadurist neid regioone kasutada. Ühe lisana on ka pärast alglaaduri koodi-, andme- ja nullidesektsiooni koht ka alglaaduri enda kontrollsumma jaoks.

5.2 Alglaaduri abiteek

Alglaaduri abiteek koosneb ühest päisfailist ja ühest C-failist, mis realiseerib päises kirjeldatud funktsioonid. Abiteek on tehtud selleks, et lihtsustada ülejäänud tarkvaral alglaaduriga suhtlemist. Alglaadur kasutab ise ka abiteeki, sest seal on funktsioone, mida peaksid kasutama nii põhitarkvara kui ka

alglaadur. Näiteks käskude loendi lugemine on tegevus, mida võiks saada teha välimine tarkvara käskude loendi kontrollimiseks ja alglaadur ise käskude täitmiseks.

Abiteegis on funktsioonid käskude loendi lugemiseks ja muutmiseks. Kuna käskude loend asub välimises FRAMis, siis on abiteegil vaja funktsioone, et FRAMi lugeda ja kirjutada. Samas on teada, et nii alglaaduris kui ka põhitarkvaras, mis mõlemad abiteeki kasutavad, on olemas juba funktsioonid selle jaoks. Seega on mõistlik kasutada juba olemasolevaid funktsioone ka abiteegis. Levinud viis on selleks kasutada funktsiooniviitasid, mis kaasa antakse, kuid nendest loobuti. Alglaaduri päises defineeriti lihtsalt funktsioonide prototüübid. Kasutades funktsiooniviitasid oleks vea korral probleem tekkinud programmi töö käigus, aga käesoleva lahenduse korral tuleb viga välja juba linkimise faasis: kui prototüüpidele vastavate funktsioonide realisatsiooni ei leita, siis annab linkija veateate.

6 Töö käik

Siin peatükis on kirjeldatud teised sarnased lahendused, töö käigus tekkinud probleemid ja alglaaduri testimine.

6.1 Sarnased lahendused

Enne koodi kirjutamist tutvuti STMicroelectronicsi tehtud näidislahenduse ja selle lähtekoodiga [11]. Samuti tutvuti ka STM32F2XX seeria jaoks tehtud näidisprojektiga [12], mis oli põhiosas samasugune eelnevaga.

Nendest lähtekoodidest võeti ka jaotises 5.1.2 kirjeldatud süsteemi päised ning sellele järnevas jaotises 5.1.3 kirjeldatud failid.

6.2 Suurimad probleemid

Siin peatükis on väljatoodud mõned enim aega kulutanud probleemid.

6.2.1 Viga päises

Mikrokontrolleri riistvaraga suhtlemine käib läbi juht- ja staatusregistrite, mis asuvad programmi aadressiruumis. Lihtsustamaks nendega suhtlemist teevad mikrokontrolleri tootjad C päisfailid, kus on iga mooduli jaoks defineeritud selle registrite aadressid. Nii ei pea igaüks uuesti oma koodis seda ise tegema, vaid kasutab olemasolevaid päisfaile. See muudab ka kirjutatava koodi loetavamaks.

Töö autor kopeeris alglaaduri jaoks need registrite nimesid ja aadresse siduvad päisfailid STMicroelectronics-i näidisprojektidest [11, 12]. Testides ja realiseerides I^2C suhtlust CAMi prototüübil, tekkis probleem, et asi lihtsalt ei töötanud. Pärast mitmeid tunde vea otsimist leiti, et sisend-/väljundviigud ei ole õigesti seadistatud: I^2C moodul ei saanud neid kasutada. Kui kõik teised veakohad olid välistatud, avastati vea tegelik põhjus. Nimelt oli viga tootja poolt tehtud päises. Viikude seadmistamisega riistvaramoodulile seotud registrile vahetult eelneva registri suurus oli päises 32 bitti, kuid manuaalis oli kirjas, et see on 16-bitine register. See tähendas, et vajalikust registrist kirjutati 16 biti võrra mööda.

6.2.2 *Chip Write Enable*

Kui I^2C suhtlus CAMi prototüübil toimima saadi ja suhtlus FRAMiga toimus, siis asuti seda sama tegema ka CDHSi plaadil. Riistvaraline I^2C moodul ise oli küll samasugune, aga sisend-/väljundviikude seadistamine oli natuke erinev. Tundus nagu asi töötaks, aga FRAMi kirjutades ja siis lugedes ei olnud tagasiloetavad andmed samad, mis sinna kirjutati.

Tagantjärele oli vea põhjus suhteliselt ilmne, kuid sellegipoolest kulus palju aega enne, kui veale jällile jõuti. Nimelt oli kasutataval FRAMi kiibil eraldi jalg, mis pidi olemas kindlas asendis, et FRAMi kirjutamine töötaks. CAMi plaadil oli see jalg jäädavalt õigesse asendis, aga CDHSi peal oli see viik ühendatud mikrokontrolleri viigu külge ja nii pidi tarkvara selle viigu enne kirjutamist FRAMi õigesse asendisse panema.

6.2.3 *SPI slave select*

Kui selgus, et on vaja CDHSi peal kasutada SPI siinil olevat FRAMi, esines ka selle realiseerimisel kaks rohkem silumiseks aega võtnud probleemi.

Esimesel katsel pärast paaritunnist katse-eksitusmeetodil üritamist avastas autor, et kasutataval prototüübil polnud SPI siinil FRAMi kiipi. Kiip oli lihtsalt trükkplaadile jootmata jäänud, kuid see ei olnud ainus probleem, mis aega võttis.

Kuna töö autorile öeldi ainult alglaaduri tööks vajaminevate asjade kohta, siis jäi SPI siinil olevate teiste seadmete *slave select* viigud seadistamata. Ostsillograafiga SPI suhtlust vaadates leiti, et tagasitulevad andmed on küll väga imelikud, kuid neis on mingi loogiline struktuur. Tuli välja, et SPI siinil olev reaalajakell üritas samal ajal koos FRAMiga andmeid tagastada. Probleemi lahendus oli jällegi lihtne: määrati kõigi SPI siinil olevate seadmete valimise viigud mitteaktiivseks. Alglaaduri töö ajal muudeti ainult FRAMiga suhtlemise ajaks FRAMile vastavat seadme valimise viigu väljundit.

6.3 Testimine

Testimiseks ja silumiseks kasutati mikrokontrolleril olevat silumise süsteemi, millega suheldi kasutades Seggeri JTAGi [13,14] emulaatorit J-Link [15]. Kuna CAMi prototüübil polnud JTAGi liidese jaoks kõik signaalid väljatoodud, kasutati seal vähemate signaalidega *Serial Wire Debug* (SWD) liidest, sest nendele signaalidele oli juurdepääs olemas.

Siluriks kasutati *The GNU Project Debbugger*it (GDBd). J-Link ja GDB võimaldasid:

- suvalises kohas mikrokontrolleri töö peatada,
- lugeda ja kirjutada SRAMi,
- lugeda ja kirjutada riistvaramoodulite juhtimiseks kasutatavatesse registritesse,
- väljakutsuda funktsioone,
- pärast muudatuste tegemist mikrokontrelli tööd jätkata pooleliolevast kohast.

Siluri abil sai simuleerida erinevaid süsteemi olekuid ja nii said kõik erinevad koodirajad vähemalt korra käivitatud.

Alglaaduri integreerimisega tegelesid CDHSi ja CAMi arendajad.

7 Teadaolevad vead

Käesoleva dokumendi kirjutamise ajal on ESTCube-1 juba Maa orbiidil ja enam ei jää üle muud, kui vead lihtsalt dokumenteerida.

7.1 Puuduoleva tarkvara tõmmise kontrollsumma arvutamine

Kui juhtub, et alglaadur tahab kontrollida tühjas välkmälu pesas oleva tarkvara tõmmise kontrollsummat, siis loetakse päisest, et tarkvara suurus on 0xFFFF FFFF baiti ja hakkab seda arvutama. Tegelikult tuleks kontrollida, et tarkvara oleks mõistlikku pikkusega. Sobivaks maksimumiks oleks näiteks tarkvarapesa maksimaalne pikkus.

Sama probleem esineb ka siis, kui enne FRAMist välkmällu kopeerimist hakatakse kontrollima FRAMis oleva tarkvara tõmmise kontrollsummat.

7.2 Südamelöögi genereerimine

Südamelöögi genereerimisel on kaks probleemi, millest esimene on lihtne lohakusviga. Nimelt jäi CAMi versioonil südamelöögi signaali genereerimiseks kasutataval funktsiooni keha tühjaks. Seega CAMi peal ei genereeri alglaadur ühtegi lööki.

Teine probleem tuleneb sellest, et lõppfaasis oli vaja CDHSi peal FRAMiga suhtlus ümber teha SPI andmesiini peale. Seetõttu jäi SPI FRAMi koodi südamelöögi signaali genereerimine lisamata. See võib olla probleem, kui kopeerimise käsu ajal kõigepealt välises FRAMis oleva tarkvara tõmmise kontrollsummat arvutatakse ja selle vältel ühtegi südamelöögi signaali ei genereerita.

Nende kahe probleemi leevendamiseks võib öelda, et alglaaduri tegemise ajal polnud toitesüsteemis südamelöögi jälgimine veel realiseeritud. Selle dokumendi kirjutamise ajal on aga selge, et see funktsionaalsus polegi vajalik. Nimelt otsustati, et tarkvara töötamisest saab paremini aru, kui moodul vastab toitesüsteemi poolt saadetud päringule. Päringu saatmise periood on piisavalt pikk, et alglaadur jõuab oma töö lõpetada ja seetõttu ei pea alglaadur ise nendele päringutele vastama.

8 Mida tulevikus paremini teha?

8.1 Programmi ettevalmistamine

Alapeatükis 3.3 kirjeldati, miks on vaja programm linkida kindlale mäluaadressile. Samas on teada, et kompilaatorid suudavad genereerida ka koodi, mis ei sõltu positsioonist. Üheks järgmiseks uurimissuunaks on kontrollida, kas see töötab nagu soovitud. Katkestusvektorite tabelis on ikkagi absoluutsed aadressid, kuid alglaadur võib tarkvara kopeerimise ajal ise need aadressid siis vastavalt programmi asukohale transleerida. Nii oleks võimalik kasutada ühte satelliidile üleslaaditud tarkvara tõmmist nii sisemise välkmälu primaarses kui ka sekundaarses pesas.

Hetkel kasutavad nii alglaadur ise, kui ka alglaaduriga suhelda tahtev programm alglaaduri abiteeki. See tähendab, et abiteek on nii alglaaduri kui ka põhitarkvara tõmmises ehk abiteek on välkmälus korduvalt. Tulevikus võib kaaluda sellist lähenemist, et abiteek on alglaaduri osa ja põhitarkvara linkimisel öeldakse, millistelt aadressidelt abiteek kätte saadakse. Sellisel juhul ei oleks vaja abiteeki mitmekordselt hoiustada.

8.2 Polsterduse eemaldamine

Alapeatükis 4.3 sai mainitud, et tarkvara tõmmise ja päise vahel on 500 baiti polsterdust, mis raiskab mäluseadmetes ruumi. Tulevikus tasub uurida, kuidas see polsterdust eemaldada.

8.3 Enne juhtimise üleandmist algseisu taastamine

Alglaadur kasutab mõnda riistvaramoodulit ja muudab selle käigus nende juhtregistrite sisu algväärtusest erinevaks. Kui alglaaditav programm eeldab, et algväärtus on kindel ja ei tee ise alguses moodulile resetti algväärtuste taastamiseks, siis programm töötab iseseisvalt, aga mitte koos alglaaduriga.

Alglaaduris tehakse enne mooduli seadistamist moodulile alglähtestamine, et olla kindel, et mooduli registrites oleksid teadaolevad algväärtused. Samamoodi tuleks kõigile kasutatavatele moodulitele teha alglähtestamine ka enne alglaaditavale programmile juhtimise üleandmist.

9 Kokkuvõte

Käesoleva töö raames loodi satelliidi ESTCube-1 kahele alamsüsteemile alglaadur, mis võimaldaks nende tarkvara töö käigus uuendada.

Alglaadurit on pidevalt tarkvara uuendamiseks kasutatud, sest pärast satelliidi komplekteerimist polnud võimalik teisel viisil nende moodulite tarkvara uuendada. Viimane uuendus tehti enne satelliidi orbiidile viimist Kourous. Orbiidil olles on alglaadur alglaadinud CDHSi tarkvara. Lähitulevikus lülitatakse sisse ka CAM. Siiamaani pole orbiidil nende alamsüsteemide tarkvara uuendatud, kuid Maa peal on seda korduvalt tehtud.

Kokkuvõtvalt said töö eesmärgid täidetud ning töö lõpp-produkt on Eesti esimesel satelliidil ESTCube-1 reaalselt kasutusel.

Bootloader for ESTCube-1 Command and Data Handling System and Camera module

Bachelor Thesis (6 ECTS)
Karl Tarbe
Summary

Estonian first satellite ESTCube-1 consists of several modules. The main on-board computer Command and Data Handling System (CDHS) and Camera module (CAM) have similar micro controller units (MCUs): CDHS uses STM32F1 and CAM STM32F2 series micro controllers manufactured by STMicroelectronics. The goal of this thesis is to design and implement a bootloader, which enables software upgrading for CDHS and CAM, while ESTCube-1 is orbiting Earth.

The bootloader does not have to communicate with ground station. It is assumed that main firmware will store new firmware image in external FRAM and then orders bootloader to copy it to MCU's internal flash memory. External FRAM is accessed either over SPI or I^2C bus.

For the main firmware to give commands to bootloader a command list is stored in external FRAM. Firmware writes commands to there and then reboots the MCU. After MCU reset, bootloader is always started. Bootloader reads the command list and executes given commands before bootloading the main firmware.

Bootloader was implemented as 2 Eclipse projects. First for the bootloader itself and second for a utility library, which the main firmware can use to modify the command list and read logs produced by bootloader.

So far bootloader has been used many times to upgrade firmware or to boot into the main firmware. Bootloader has also successfully booted CDHS while ESTCube-1 is on low Earth orbit. While the satellite has been orbiting Earth, there has been no firmware upgrades yet.

In conclusion the goals of this thesis were achieved and the end-product is actually used on the first Estonian satellite ESTCube-1.

Tänusõnad

Indrek Sünter oli see inimene, kes oli autori kontaktisik satelliidi meeskonnas. Tema tegeles CDHSi põhitarkvara kirjutamisega ja seega oli tema põhiline ,,klient", kelle vajadusi alglaadur rahuldama pidi. Ta aitas arutada ja nõudeid paika panna, samuti luges ta elektriskeemidelt ette vajalikud sisend-/väljundviigud ning aitas prototüüpide ühendamisel. Suured tänud talle!

Lisaks saab tänu osaliseks autori hea sõber Mihkel Heidelberg, kes autorile selle teema soovitas.

Viited

[1] CubeSat Design Specification. `http://www.cubesat.org/images/developers/cds_rev12.pdf`. Revision 12.

[2] Priit Ennet Silver Lätt. Tudengisatelliit katsetab elektrilist päikesepurje. `http://teadus.err.ee/artikkel?cat=1&id=355`, aprill 2009.

[3] Indrek Sünter. *Radiation tolerant hardware design for ESTCube-1 Command and Data Handling System*. Bakalaureuse töö, Tartu Ülikool, 2011.

[4] Henri Kuuste. *ESTCube-1 tether end mass imaging system design and assembly*. Bakalaureuse töö, Tartu Ülikool, 2012.

[5] STMicroelectronics. *PM0059: STM32F205xx, STM32F207xx, STM32F215xx, STM32F217xx, Flash programming manual*, mai 2011. Revision 4.

[6] STMicroelectronics. *PM0068: STM32F10xxx XL-density Flash programming*, jaanuar 2012. Revision 2.

[7] Fujitsu. Fram technology backgrounder. `http://www.fujitsu.com/emea/services/microelectronics/fram/technology/`.

[8] STMicroelectronics. *RM0008: STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs*, oktoober 2011. Revision 14.

[9] STMicroelectronics. *RM0033: STM32F205xx, STM32F207xx, STM32F215xx and STM32F217xx advanced ARM-based 32-bit MCUs*, detsember 2011. Revision 4.

[10] STMicroelectronics. *PM0056: STM32F10xxx/20xxx/21xxx/L1xxxx Cortex-M3 programming manual*, märts 2011. Revision 4.

[11] STMicroelectronics. *AN2557: STM32F10x in-application programming using the USART*, oktoober 2010. Revision 8.

[12] STMicroelectronics. *AN3374: STM32F2xx in-application programming using the USART*, aprill 2011. Revision 1.

[13] Ieee standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture. *IEEE Std 1149.7-2009*, pages c1–985, 2010.

[14] Wikipedia. Joint test action group. http://en.wikipedia.org/wiki/Joint_Test_Action_Group.

[15] SEGGER Microcontroller. J-link. http://www.segger.com/jlink.html.

Lisa 1

Käesoleva töö käigus valminud alglaaduri ja selle abiteegi lähtekoodid on kättesaadavad ESTCube-1 meeskonnalt. Alternatiivselt on lähtekood üleslaetud ka Tartu Ülikooli Arvutiteaduse instituudi lõputööde registrisse käesoleva bakalaureuse töö lisana. Register on asub aadressil http://comserv.cs.ut.ee/forms/ati_report/index.php .

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Karl Tarbe (sünnikuupäev: 20.02.1991)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose ,,Alglaadur ESTCube-1 käsu- ja andmehaldussüsteemile ja kaamera-moodulile",
   mille juhendaja on Meelis Roos,

   (a) reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   (b) üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **07.05.2013**