TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Thomas Johann Seebecki elektroonikainstituut

Kaarel Vandler 142947IALB

HELISÜNTESAATOR ROBOTI VÄLJATÖÖTAMINE

Bakalaureusetöö

Juhendaja: Priit Ruberg
MSc
Nooremteadur

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kaarel Vandler

22.05.2017

Annotatsioon

Antud töö eesmärgiks oli arendada prototüübid 8-bitilisest süntesaator robotist ning selle juhtimiseks mõeldud juhtpuldist. Mõlemale seadmele disainiti trükkplaadid ning komplekteerimine toimus käsitsi jootes. Töö valmis ligikaudu üheksa kuu jooksul.

Heli genereerimiseks vajalik sisend loetakse aluspinnal asuvatelt triipudelt valgussensoritega, mis asuvad roboti alumisel küljel. Kogu juhtloogika, sealhulgas helisignaali genereerimine, toimub FPGAs ning riistvarakirjelduskeeleks on kasutatud Verilogi. Töö käigus valmis nii robotist, kui ka juhtpuldist töötav prototüüp ning selle edasiarendus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 23 leheküljel, 6 peatükki, 36 joonist, 2 tabelit.

Abstract

Development of sound generator robot

The main purpose of the thesis was to develop prototypes for an 8-bit sound generator robot and a remote controller for the robot. Printed circuit boards were designed and they were assembled by manually soldering the components. Development of the prototypes was done in nine months.

Input for generating the sound signal was read by photosensors from white stripes on the ground. Every photosensor corresponds to one sound frequency. The robot could generate polyphonic sound signal if more than one sensor was triggered.

Data processing and the signal generating was done by FPGA and the used hardware description language was Verilog. In the final version of the code there was used thirteen inputs and five outputs. In the top-level module there are six sub-modules being used. About six hundred lines of code were written during the development.

The remote controller communicates with the robot over radio. User interface of the remote is four push-buttons. Combination of the buttons determine rotation speed and direction of the servomotors on the robot.

During the development, there was designed two revisions of prototypes of the sound generator robot and the remote controller. The first revision had some major electronic design errors in FPGA configuration nets. This and some other minor errors were fixed in the second revision of the devices. There are also many possibilities in further development of the robot and the remote controller.

The thesis is in Estonian and contains 23 pages of text, 6 chapters, 36 figures, 2 tables.

Lühendite ja mõistete sõnastik

FPGA	*Field-programmable gate array,* väliprogrammeeritav ventiilmaatriks
BGA	*Ball grid array*, pallivõre korpus
MOSFET	*Metal-oxyde-semiconductor field-effect transistor*, metall-oksiid-pooljuht väljatransistor
LiPo	Liitiumpolümeer
TQFP	*Thin quad flat pack,* õhuke neljapandiline pisikorpus
LDO	*Low Drop-Out regulator*, lineaarne pingestabilisaator

Sisukord

Jooniste loetelu

Tabelite loetelu

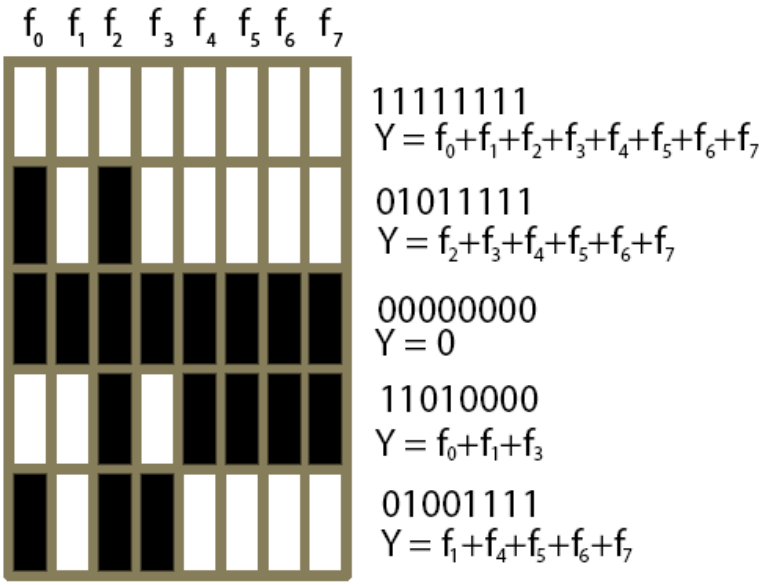
1 Sissejuhatus

Robootika teerajajaks võib nimetada Kreeka füüsikut Ctesibiust, kes aastal ca 250 eKr valmistas ajamõõtja, mis põhines vee voolamisel [1]. Sellest ajast saati on tehtud masinaid, mis aitavad inimesi erinevatel töödel. Esimese elektroonilise muusikainstrumendi loojaks võib aga pidada USA leiutajat Elisha Grayd, kes disainis telegraafi multiplekseri, mille tööpõhimõtteks oli erinevate sagedustega signaalide edastamine liinil samaaegselt, võimaldades edastada korraga erinevaid sõnumeid [2]. Ka käesolev töö seob omavahel kaks kirjeldatud teemat: robootika ja elektroonilised muusikainstrumendid.

Antud bakalaureusetöö eesmärgiks on luua 8-bitise heli süntesaator-robot, mida juhitakse kaugjuhtimispuldiga. Töö käsitleb heli, sealhulgas ka polüfoonilise heli, sünteesimise ja genereerimise meetodeid. Lisaks kirjeldatakse vastava elektroonika ja programmikoodi tööpõhimõtteid. Töö viidi läbi kahes versioonis: esialgne prototüüp ning selle edasiarendus. Disainitud trükkplaadid komplekteeriti ning joodeti autori poolt käsitsi. Ülesanne püstitati autori tööandja-ettevõtte Krakul OÜ poolt.

1.1 Ülesande püstitus

Projekteerida, disainida ja komplekteerida trükkplaat seadmele, mis suudab liikudes genereerida vähemalt 8-bitist polüfoonilist heli, mille signaal on genereeritud loogikalülituste abil. Loodaval seadmel olgu vähemalt kaks eraldiseisvat kaugjuhitavat elektrimootoritel ratast. Heli genereeritakse seadme alumisel küljel olevate valgussensorite abil, lugedes valgeid triipe, kus igale sensorile vastab üks helisagedus (1 bitt) (Joonis 1). Seade saab elektrienergiat üheelemendilisest liitiumpolümeerakust. Loogikalülitusteks kasutada tootja *Altera* väliprogrammeeritavat ventiilmaatrksit ning eelistada riistvarakirjelduskeelena Verilogi. Trükkplaadi maksimaalseteks mõõtmeteks olgu 120mm x 120mm ning see peab mahutama endal kõiki kirjeldatud komponente.



Joonis 1. Polüfoonilise helisignaali väljund sõltuvalt tuvastatud joontest.

Mootorite kaugjuhtimiseks luua eraldi kaugjuhtimispult, mis kommunikeerub helisüntesaatorseadmega raadio teel. Raadiosuhtluseks kasutada *RoC*-i. Kaugujuhtimispuldil olgu vähemalt 4 surunuppu, millega antakse juhtimiseks vajalikku sisendinfot. Toite saamiseks kasutada sarnast üheelemendilisest liitiumpolümeerakut. Puldile disainida trükkplaat, mille maksimaalseks mõõtmeks olgu 60mm x 60mm.

Elektroonika arendamisel sekundaarseks nõudeks on hoida seadme lõpp-maksumus minimaalne. Komponentide valikul eelistada pindmontaažkomponente, kuid vältida *BGA* (*Ball grid array*, pallivõre korpus) pakendeid, kuna nende käsitsi jootmine on keeruline. Takistite ja keraamiliste mitmekihiliste kondensaatorite kasutamisel eelistada võimalusel tollmõõdustiku 0603 pakendeid. Pistikute ja muude suuremate komponentide puhul on sobiv ka aukmontaažkomponentide kasutamine.

Komponentide valikul ning skeemi disainimisel tuleks arvestada ka ettevõtte standarditega. Elektroonika disainimiseks kasutada programmi *Altium Designer* 16. Võimaluse korral eelistada sümbolite ja komponentide jooteväljade mustrite kasutamist ettevõtte teegist.

2 Komponentide valik

Komponentide valikul lähtuti ülesande püstitusest, seadme lõppmaksumusest ning pakendite(korpuste) tüübist. Eesmärk oli saavutada tulemus võimalikult soodsalt, kuid ka lihtsalt. Kuna jootmine toimus käsitsi, siis välditi *BGA* tüüpi korpuste kasutamist.

2.1 FPGA ja ostsillaatori valik

Kuna ettevõttes oli eelnevalt olemas tootja *Altera* FPGAde (field-programmable gate array, väliprogrammeeritav ventiilmaatriks) konfigureerimist võimaldav programmaator, siis eelistatigi nimetatud tootja FPGA kasutamist. Eelnevalt on välja arvestatud, et kasutatavate sisend- ja väljundviikude arv ei ole suurem kui 20 (Tabel 1).

Tabel 1. FPGA väljaviikude arvestus funktsiooni kohta.

Sisend-väljundseadme funktsioon	**Kasutatav väljaviikude arv**
Valgusmõõtjad	9
Servomootorite juhtsignaal	2
Heliväljund	1
Juhtsignaalid puldist	4
Valgusdioodid	4
KOKKU	20

Antud tingimustel tundus mõistlikuim kasutada *Altera Cyclone* seeria FPGAd, kuna arenduse hetkel oli tegemist ainukese tooteseeriaga, kus oli lisaks BGA tüüpi korpustele ka TQFP (*thin quad flat pack*, õhuke neljapandiline pisikorpus) tüüpi korpustega integraalskeeme ning selle tootekirjelduses oli välja toodud, et see on mõeldud madala võimsusega ja hinnatundlikule rakendusele [3]. Antud tooteseeria seast valiti aastal 2002 alustatud alaseeria *Cyclone* seade EP1C3T100C8N, kuna see oli antud seerias minimaalseima sisend- ja väljundviikude arvuga, milleks on 65, ning selle pakendiks on TQFP. [4]

EP1C3T100C8N maksimaalne toetatav töösagedus on 405 MHz. Takteerimissignaali sageduseks valiti tööandja soovitusel 25MHz ning taktsignaaligeneraatoriks otsustati kasutada välist kristallostsillaatorit, kuna võrreldes näiteks operatsioonivõimenditel põhinevate ostsillaatoritega, vajab kristallostsillaator vähem komponente ning on skeemi keerukuselt lihtsam. Kuna FPGAl taktsignaali sisendviigul olev mahtuvus on 4.7 pF, siis ostsillaatoriks valiti FOX ELECTRONICS FXO-HC736R-25, mille maksimaalne mahtuvuslik koormus võib olla kuni 15 pF [5] [6].

2.2 Mäluseadme valik

Kuna antud FPGA ei oma endas mittevolatiivset mälu, siis tuli see lisada väliselt, kuna antud FPGA nõuab konfigureerimist igal käivitumisel. Altera on soovitanud enda FPGAdel kasutada tootjapoolset *flash* mäluseadmeid, mida nimetatakse konfiguratsiooniseadmeks. Pärast FPGA toite sisselülitamist jääb seade ootama konfiguratsiooniinfot loogikalülituste kohta. Antud spetsiaalne mäluseade ongi mõeldud Altera FPGAde sisselülitamisel konfiguratsiooniandmete nõudmisel selle esitamiseks. Lisaks on antud integraalskeemi kasutamine ja programmeerimine toetatud ka Altera tarkvaraga [7].

Ülesande püstitusel eeldati, et HDL kood ei ole väga keerukas ning pikk, siis võib ka eeldada, et mälumaht ei pea olema väga suur. Selle tõttu valiti mäluseadmeks seeria odavaim toode EPCS1SI8N, mille mälumahuks on 1 Mbit.

2.3 Pistikute valik

Pistikute valikul lähtuti suures osas ettevõtte standardist kasutada *Hirose* DF11 pistikuid. Nimetatud pistikuid kasutati mõlemas versioonis nii aku, valjuhääldi, servomootorite ning raadiomooduli programmaatori ühendamiseks. Kuna eelistati kasutada võimalikult palju ühesuguseid pistikuid, siis esimeses variandis kasutati kuue viiguga pistikuid. Teises versioonis kasutati nelja viiguga pistikuid.

FPGA konfigureerimine toimub spetsiaalse programmaatoriga *Altera USB-Blaster*, mille liideseks on JTAG. Antud seadmel on konfigureerimispistikuna kasutatud 2,54mm sammuga 10-viigulist pesa. Sellest tulenevalt on ka FPGA konfigureerimispistikuna kasutatud samasugust pesa.

2.4 Mootorite ja juhtmevaba andmeedastusmoodul

Mootorite juhtimine peaks toimuma FPGAlt tulnud signaalidega. Selleks oleks tarvilik lisaks disainida mootorite juhtplokk või kasutada muud sarnase otstarbega integraalskeemi. Lahenduseks kasutati modifitseeritud servomootorit, kus juhtsignaaliga ei tüürita mitte mootori oleku nurka vaid kiirust. Antud modifikatsioon kujutas endas servomootorite ülekandehammasrataste ringi käimist taktistava mehhaanilise stopperi eemaldamist ning elektroonilise tagasiside potentsiomeetri asendamist pingejaguriga (kaks 10 kΩ takistit) [8].

Kuna seada töötab akutoitel, siis otsustati kasutada mootorit, mis on võimalikult väikese voolutarbega, kuid piisavalt võimas, et antud seadet vedada. Kuigi sobilikke sarnaste parameetritega servomootoreid on mitmeid, siis tööandja soovitusel valiti välja tootja Turnigy analoogservomootor TG9e [9]. Selle tööpingeks on 4.8V ja juhtsignaaliks on pulsi pikkuse modulatsiooniga(*Pulse Width Modulation, PWM*) tüüritav signaal.

Antud seadet juhitakse puldiga, mille omavaheline suhtlus peab toimima juhtmevabalt. Parimaks lahenduseks peeti *ZigBee* protokollil töötavaid tootja Digi International tooteseeria xBee mooduleid. XBeed eelistati selle tõttu, et nende kasutamine on mugav ning konfigureerimine on lihtne. Käesolevas töös kasutati Digi International xBee-PRO ZB (S2C) mooduleid.

2.5 Valgussensorite ja komparaatorite valik

Seade on mõeldud valgete joonte eristamiseks mustalt taustalt. Antud lahenduses kasutati peegelduslikku fotoelektrilist sensorit, mille tootjaks on OMRON ELECTRONIC COMPONENTS ja mudeliks EESY171 [10]. Antud sensoril on ühes pakendis infrapuna LED ja NPN tüüpi bipolaarne fototransistor. Seadmes kasutati üheksa eelnevalt mainitud sensorit.

Kuna EESY171 tüüpi valgussensori väljund on analoogsignaal ning antud FPGAl pole analoogsisendeid, on vaja signaal digitaalseks muundada. Analoog-digitaalmuundurina kasutati analoogkomparaatoreid operatsioonivõimendi põhjal. Alternatiivselt oleks võinud kasutada mõnda analoog-digitaalmuunduri integraalskeemi, kuid nende väljund vastab üldiselt mõnele digitaalsele suhtlusprotokollile ning sellele oleks pidanud kirjutama ka liidese FPGAle. Liigse keerukuse tõttu välditi analoog-digitaalmuundurite

integraaalskeemide kasutamist. Antud juhul kasutati iga sensori kohta üks komparaator, et saada üks bitt informatsiooni iga sensori kohta: kas antud sensori all on valge triip või mitte. Tulemusena saab sensoritelt edastada info üheksa bitise vektorina.

Kuna oluline oli ka hind ning füüsilised mõõtmed, siis kasutati integraalskeeme, kus ühes pakendis on kaks komparaatorit. Kasutati STMICROELECTRONICSi toodetud LM2903DT mikroskeemi [11].

2.6 Audiovõimendi ning valjuhääldi valik

Arvestades, et kasutatava FPGA maksimaalne väljundvool on 25 mA ühe väljaviigu kohta [12], siis on vaja kasutada võimendit. Kuna väljundpinge 3,3V on sobilik, siis sobis võimendus ainult voolu järgi.

Versioonis 1 kasutati selleks tootja STMICROELECTRONICS operatsioonivõimendi baasil audiovõimendit TS4871ID [13]. Versioonis 2 asendati võimendi integraalskeem indutseeritud N-kanali väljatransistoriga, mida tüüriti FPGA pealt tulnud pingega paisu peale.

Seadme esimeses versioonis ei pandud väga suurt rõhku valjuhääldi füüsilistele mõõtmetele ja raskusele, vaid et koormuselt sobiks võimendiga ning, et kõlari mähis taluks vastavat voolu. Kuna audiovõimendi TS4871ID võimsus on 1W 8 Ω koormusega 5-voldise toitepinge juures.Esialgu kasutati välist kõlarit, mis oli 8 Ω takistusega ning 1 W võimsusega, ning see sobis hästi testimiseks. Kuna antud kõlar oli väline, siis ühendati see trükkpaadiga juhtmetega vastavast pistikust.

Teises versioonis arvestati rohkem seadme mobiilsusega ning sellega, et pole enam vajadust valjuhääldit pidevalt teisaldada trükklaadilt. Sellest tulenevalt kasutati pindliitekomponendi(*Surface-Mounted Device, SMD*) valjuhääldit, mis joodeti otse trükkplaadile. Antud juhul valiti tootja Mallory Sonalert valjuhääldi PSR1511N08S3.5K [14]. Nimetatud kõlarielemendi nimivõimsus on 0.7 W ning takistuseks 8 Ω.

3 Elektroonika

Elektroonika disainimiseks kasutati programmi *Altium Designer* 16. Kogu elektriskeem on saadaval Lisas 2. Seadmetest genereeriti nimetatud programmi abil ka 3D mudelid, mis on saadaval Lisas 2. Valminud jooniste põhjal genereeriti ka tehasele sobivad failid trükkplaadi tootmise jaoks. Täielikud elektri- ja trükkplaadiskeemid on saadaval Lisas 2.

3.1 Toited

Kuna roboti kui ka puldi seadmetes kasutatakse erinevaid integraalskeeme, on vaja erinevaid toitepingete nivoosid. Energiasäästlikuim variant oleks kasutada kõikide vajalike pingenivoode saavutamiseks impulsspingeregulaatoreid, mida on üpriski keeruline disainida. Töö üleliigse keerukuse vähendamiseks on eelistatud LDO (*Low Drop-Out regulator*, lineaarne pingestabilisaator) tüüpi pingeregulaatoreid, kuid servomootorite jaoks on vaja akupingest kõrgemat pinget (5V) ning selle tõttu on kasutusel ka impulssregulaator.

Robotil on kolm LDO pingeregulaatorit: kaks 3,3 V regulaatorit ja üks 1,5 V regulaator. 1,5 V regulaator on mõeldud ainult FPGAle toite andmiseks. Selle sisend- ja väljundkondensaatorid, mis tegelevad pinge silumisega, on mahtuvusega 1 µF. 3,3 V regulaatoritest üks toidab ainult raadiosüsteemikiipi xBeed ja teine regulaator FPGA sisend-väljund süsteeme ning teisi integraalskeeme. Nende regulaatorite sisend- ja väljundkondensaatorid on mahtuvusega 2,2 µF, mis on piisavalt suured, et siluda hetkelisi pingelangusid toitesiinidel.

3.1.1 Servomootorite toide

Servomootorite Turnigy TG9e ametliku edasimüüja kodulehel on märgitud toitepinge vahemikuks 4.8 V – 6 V [15]. Kuna ei õnnestutud leida antud mootorite voolutarvet, siis tuli vastavad voolumõõtmised teha. Toitepingeks valiti 5 V ning toiteplokiga ühendati jadamisi multimeeter *Keysight* U1233A [16], millega mõõdeti voolu. Servomootorile vajalik juhtsignaal genereeriti mikrokontrolleriga Arduiono Leonardo ning selleks vajalik programmikood on avalik. Mõõtmised tehti viie minuti jooksul ning kasutati multimeetri

funktsiooni, mis registreerib minimaalse ja maksimaalse mõõtmise. Mõõdeti voolutarbimist juhtsignaali puudumisel, mootori vabal koormamata liikumisel ning liikuva mootori koormamisel takistades seda käega. Mõõtmistulemused on Tabelis 2.

Tabel 2. Servomootori voolutarbiminme.

	Väikseim mõõdetud vool	Suurim mõõdetud vool
Sisendsignaali puudumisel	3,12 mA	5,48 mA
Mittekoormatud liikuv mootor	17,5 mA	576 mA
Koormatud liikuv mootor	43,0 mA	1,12 A

Suurimaks mõõdetud tulemuseks saadi 1,12 A. Arvestades, et robotile tuleb selliseid mootoreid kaks tükki, siis halvimal juhul võib olla mootorite koguvoolutarbimine kaks korda suurem. Lisaks peab arvestama mõõteveaga 1% lühiajaliste veel suuremate voolutarbimistega, mis võivad tekkida mootorite pöörlemissuuna ning kiiruse muutumisel.

Kuna mõõtmised teostati 5 V juures, siis servomootorite pingeregulaatori väljundiks valiti 5V. Regulaatoriks valiti tootja *Analog Devices* integraalskeem ADP1614ACPZ-1.3-R7 [17], mis on lülituslik pinget tõstev regulaator. Selle väljundpinge on seatav kuni 20 V-ni ning maksimaalne väljundvool on piiratud 4 A-ga. Arvestades teostatud mõõtmisi, siis see regulaator sobib hästi antud rakendusse.

Väljundpinge seati tagasisidetakistitega R29 ja R31 (Joonis 2) ning nende väärtused on määratud valemiga (1) [17].

$$R1 = R2 * \left(\frac{Vout - 1.245}{1.245}\right) \quad (1)$$

Valemis (1) tähistatud R1 vastab Joonisel 2 takistile R31 ja R2 vastab takistile R29. Määrates takisti R29 väärtuseks 24 kΩ, saame R31 värtuseks 27.4 Ω, kuid võttes sellele lähima E24 rea takisti, saame selleks 75 kΩ. Induktori valiku puhul lähtuti komponendi füüsilisest mõõtmest, ning valiti väikseima soovitusliku väärtusega induktor, milleks on 4,7 µH. Antud disainis kasutati induktorit (L1) MGV06034R7M-10. Sellisel juhul määrab impulssvoolu induktoril valem (2).

$$\Delta I_L = \frac{Vin * t_{on}}{L} \quad (2)$$

kus

$$t_{on} = \frac{\frac{Vout - Vin}{Vout}}{f_{sw}} . \quad (3)$$

Lülitussagedus f_{sw} on 1,3 MHz.

Antud juhul tuleb impulssvooluks 0.52 A. Kui arvestada induktori nominaalseks vooluks In = (Vout/Vin)/Iout [18], milleks tuleb 5,4 A. Lisades sellele pool impulssvoolust, saame 5,66 A. Valitud induktori maksimaalne RMS vool on 5,5 A, küll aga küllastusvool on 10 A. Muidugi oleks pidanud valima suurema maksimaalse RMS vooluga induktori, kuid esialgses disainis tehti arvutusviga ning saadi tulemuseks madalam vool.

Alaldusdioodiks (D4) valiti Schottky tüüpi diood SS54. Antud dioodil maksimaalne keskmine pärivool on 5 A, maksimaalne lühiajaline vool 150 A ning maksimaalne RMS pinge 28 V. Selliste parameetritega diood sobib antud rakendusse hästi.

Kompensatsiooniskeemis tuleb arvestada tagasisideahela ristumissagedusega ning peab jälgima, et see esineks viis korda madalamal sagedusel kui

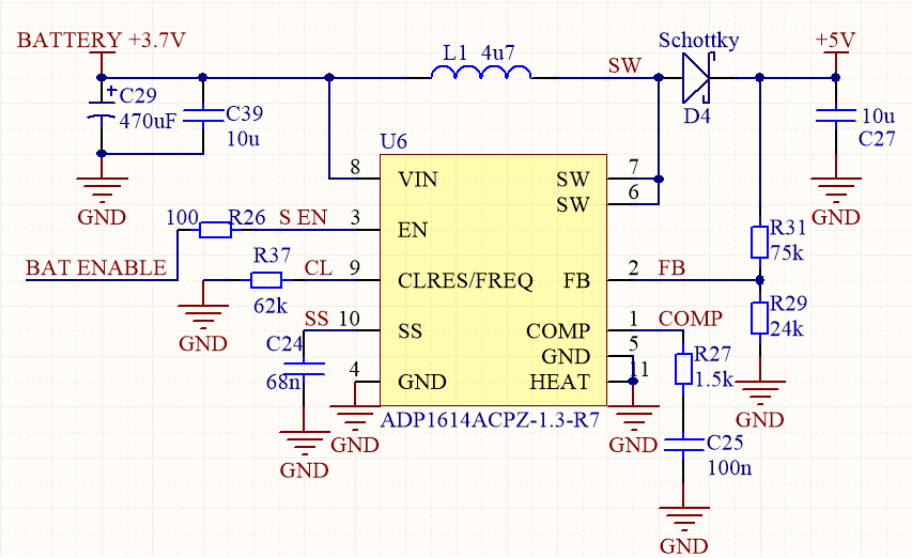
$$Fz = \left(\frac{Vin}{Vout}\right)^2 * \frac{R}{2\pi L} \quad (4)$$

ning selleks tuli Fz = 23,2 kHz. Järelikult ristumissagedus on Fc = 4,63 kHz. Sellest omakorda saab leida kompensatsiooniahela takisti (R27) ja kondensaatori (C25) väärtused, mis on määratud järgmiste valemitega:

$$R_{comp} = \frac{4806 * Fc * Cout * {V_{out}}^2}{V_{in}} \quad (5)$$

$$C_{comp} = \frac{2}{\pi * F_c * R_{comp}} \quad (6)$$

Nendeks tulevad vastavalt 1,5 kΩ ja 91 nF, kuid lihtsuse ning parema kättesaadavuse tõttu kasutame skeemis 100 nF kondensaatorit. Takistiga R37 määrati voolupiirang 4 A juurde.

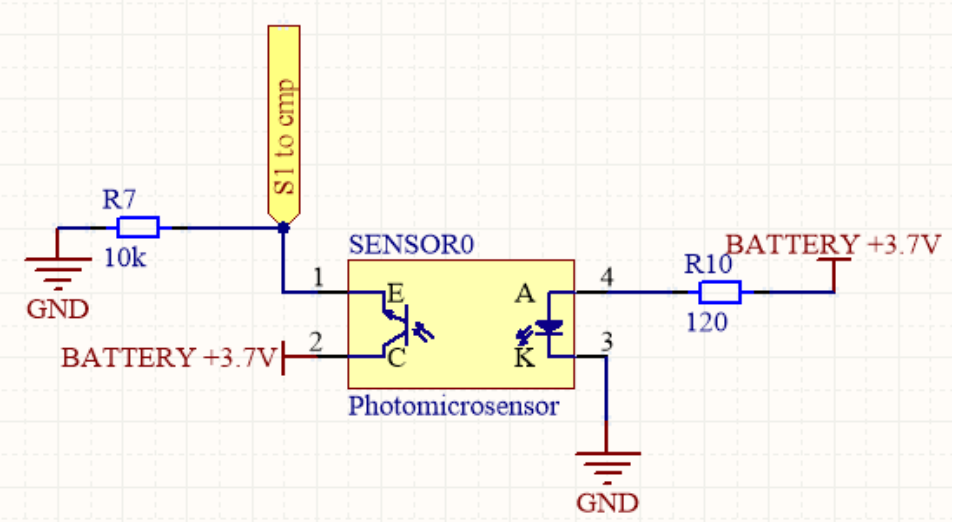


Joonis 2. Servomootorite 5 V pingeregulaator.

3.2 Valgussensorid ning joonetuvastus

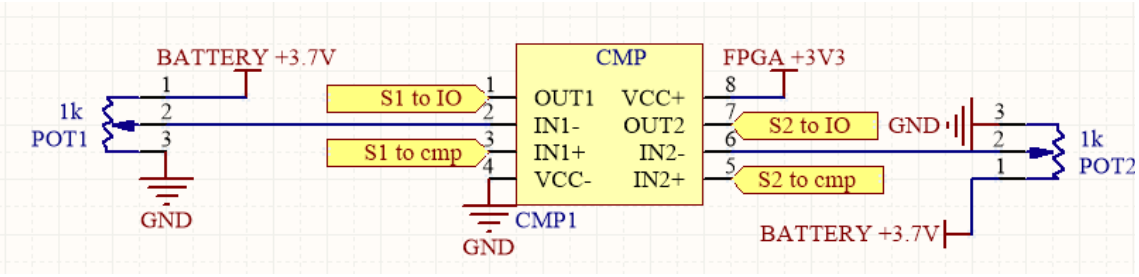
Valgussensorite LEDidega ühendati jadamisi 120 Ω takistid, et laetud aku juures ei läbiks valgusdioode liiga suur vool. Kui antud LEDidel tekkiv pingelang on 1,2 V ja soovituslik vool on 30 mA ning täis aku pinge on 4,2 V, siis takistile jääv maksimaalne pingelang on 4,2 V – 1,2 V = 3 V. Minimaalse takisti väärtuse saab Ohmi seadusest: R = U/I, 30 mA/3 V = 100 Ω. Kuigi arvutuslikult saadi takisti väärtuseks 100 Ω, kasutatakse antud lahenduses 120 Ω takistit (R3 Joonisel 3), et vältida LEDi läbilööki. Kasutades suurema väärtusega takistit, väheneb küll eralduva valguse hulk, aga arvestades, et sensor on aluspinnalt ainult mõne millimeetri kõrgusel, siis ei oma see erilist mõju joonetuvastusel.

Fototransistori kollektor ühendati otse akuga. Emitterist läheb vool läbi piirava takisti R7 maandusesse. Analoogsignaal valgustugevuse kohta võetigi nimetatud voolu piiravalt takistilt. Kuna tegemist on NPN tüüpi fototransistoriga, siis pinge takistil on võrdeline valgustugevusega: mida suurem on valgustugevus, seda suurem on pinge takistil.



Joonis 3. Valgussensori ahel.

Kuna kasutataval FPGAl pole analoogsisendeid, tuleb antud analoogsignaal diskreetida. Piisab ainult kahest diskreedist, sest vajalik on ainult valge joone olemasolu tuvastamine: „0“ – joont ei tuvastatud, „1“ – joon on tuvastatud. Probleem lahendati analoogkomparaatoritega (Joonis 4). Mitteinverteerivasse sisendisse tuleb valgussensori analoogsignaal, inverteerivasse sisendisse etalonpinge, mida määratakse potentsiomeetriga (POT1 ja POT2). Esimeses lahenduses oli antud referentsi määramiseks kasutatud potentsiomeetreid, kuid teises versioonis asendati need takistitest pingejaguritega. Kui sensori signaalipinge ületab referentspinget, on komparaatori väljundis „1“, kui signaali pinge jääb allapoole pingereferentsi, on väljundis „0“. Komparaatori väljundid on ühendatud otse FPGA sisenditesse.

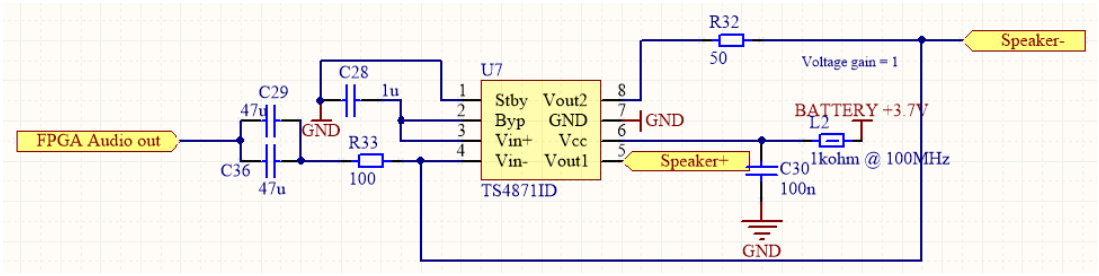


Joonis 4. Analoogkomparaator sensori signaali diskreetimiseks.

3.3 Helivõimendi

Joonisel 5 on kujutatud võimendiskeemi ühendusi. Kuna FPGA pingeväljund on 3,3 V, siis eeldati, et võimendust läheb vaja ainult voolu järgi. Selleks seati võimendustegur 1

tagasisidetakistiga R32 50 Ω ja sisendtakistiga R33 100 Ω. Sisend-kõrgpääsfilter lõikesagedusega ligikaudu 16 Hz on saadud kombineerides sisendtakisti R33 jadamisi sisendkondensaatoritega C29 ja C36 kogumahtuvusel 94 µF [19]. Võimendi integraalskeemi toitesisendi ette pandi ka ferriit L2, mis filtreerib mootoritest tekkivat müra toitesiinil.



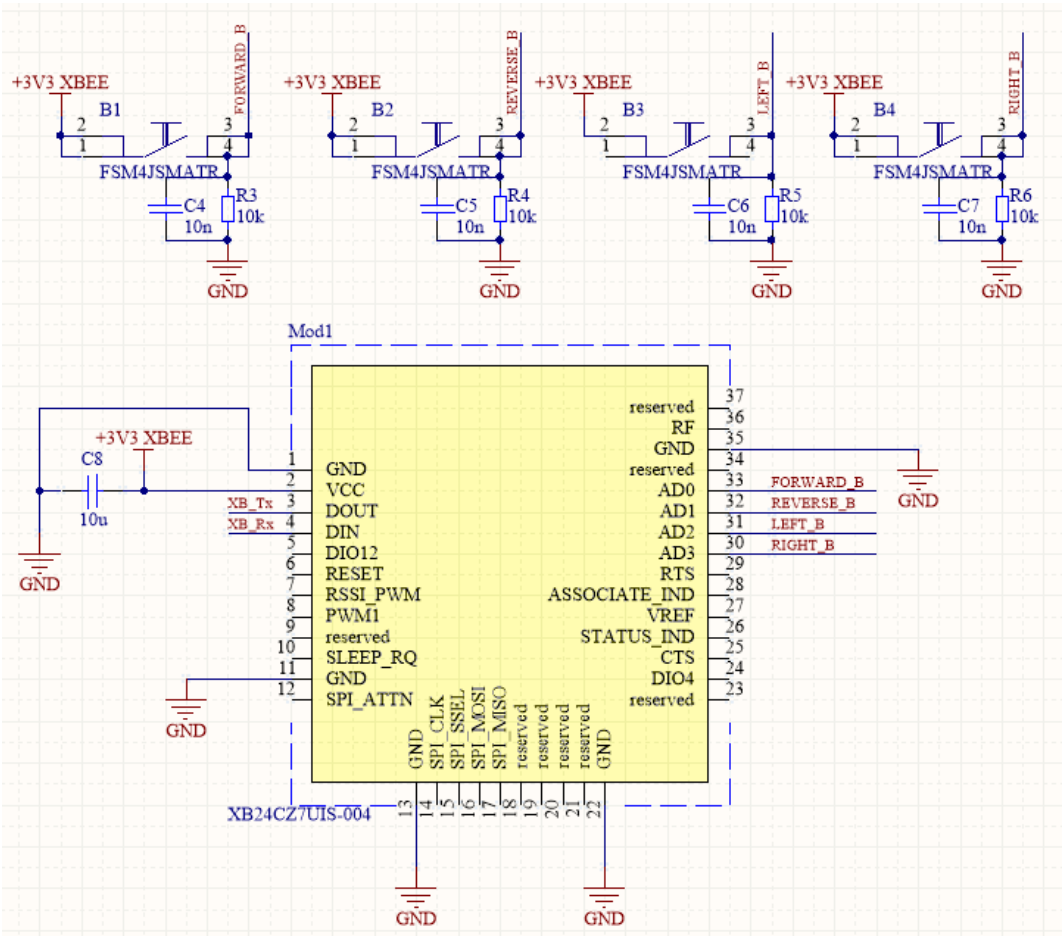
Joonis 5. Võimendiskeem.

On teada, et Joonisel 5 oleval skeemil esinevad vead, nt: tagasisidetakisti R32 peaks asetsema tagasisidesiinil pärast ühendust *Speaker*- ning kõlari ühendus peaks minema otse võimendi intergaalskeemi väljundist (väljaviik nr 8).

3.4 Raadioside ning pult

Raadioside puldi ja roboti vahel on loodud süsteemikiibi xBee abil, mis asub mõlemal seadmel. Tööpõhimõte seisneb selles, et kui puldil asuva raadioseadme väljaviigul rakendatakse loogiline „1“, siis robotil asuva seadmel on sama väljaviigu väljundis loogiline „1“. Sama kehtib ka loogilise „0“ puhul. Puldil on sisendiks nupud B1-B4, robotil läheb süsteemikiibi väljundist signaal otse FPGA sisendisse (Joonis 6). Süsteemikiipi konfigureeriti arvutiga UART protokolli kasutades.

Puldi kasutajaliideseks ongi need neli surunuppu, mille vajutamisel ühendatakse süsteemikiibi vastav sisend loogilisele „1“-le (3,3V). Kui surunupp ei ole alla vajutatud, (ehk on lahti ühendatud) siis on sisendis loogiline „0“, mis tuleb sellest, et surunupuga on paralleelselt ühendatud 10 kΩ takisti (R3 – R6) 0V siinile (Joonis 6). Antud takisti tagab ka selle, et sisendi väärtus oleks alati määratud. Nimetatud takistiga on lisaks omakorda paralleelselt ühendatud ka 10 nF kondensaator (C4-C7), et vähendada tahtmatute lülituste ja müra tuvastamist.



Joonis 6. Skeem puldipoolsest raadiokiibist koos surunuppudega.

Lisaks süsteemikiibile paigaldati puldile ka mikrokontroller *Texas Instruments MSP430G2353IPW20* [20]. Mikrokontrolleri vajalikkus on küll sekundaarne, sest seade töötaks ka ilma selleta, kuid mikrokontrolleriga saaks vajadusel lisada funktsioone ning eelprogrammeeritud juhtprogramme. Lisaks on kõikide nuppude signaalid paralleelselt viidud ka mikrokontrolleri sisenditesse ning andmevahetus süsteemikiibiga toimub läbi UART protokolli. Kasutatud kvartsresonaator (Q1) MS3V-T1R on võetud näitena kasutatud sama mikrokontrolleri arendusplaadilt. [21]

3.5 Trükkplaat

Mõlemal versioonil määras trükkplaadi laiuse valgussensorite vahekaugus teineteisest, kui ka ülesande püstituses välja toodud plaadi maksimaalsed dimensioonid. Disainitud trükkplaadid olid mõlemad kahekihilised. Esialgses versioonis oli plaadi mõõtmeteks 107,8 mm x 70 mm, teises verisoonis oli plaadi mõõtmeteks 89 mm x 113,6 mm.

Trükkplaadi disainimisel arvestati komponentide paigutusel suuresti sellega, et kiireimad signaalirajad (nt: FPGA konfiguratsioonisiinid, taktigeneraator) oleksid võimalikult minimaalse pikkusega. Lisaks jälgiti, et 5 V pingeregulaatori lülituslik võrgusõlm ei oleks tähtsate andmesiinidele liiga lähedal, sest seal võivad järsud pingeimpulsid tekitada ülekostel liigest müra. Kondensaatorite paigutamisel arvestati voolude liikumisega ning, et iga filtreeritav sõlm läbiks esmalt kondensaatorit. Selleks paigutatigi need integraalskeemidele vastavate väljaviikudele võimalikult lähedale.

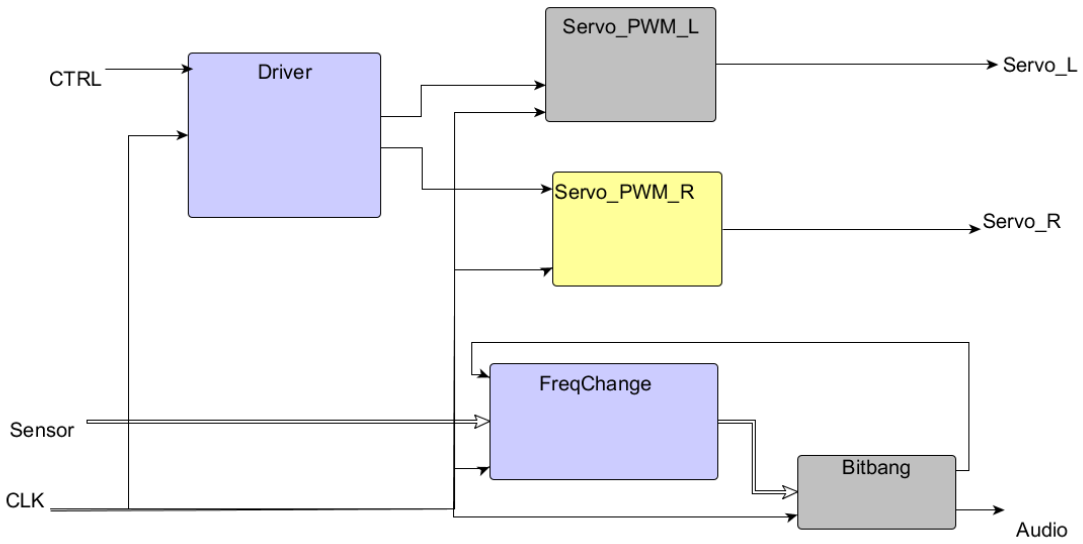
Signaalide radade laiuseks on 0.25 mm ja toitesiinide radade laiuseks on 0.5 mm – 0.8 mm. Arvestati voolude tugevusega ja suurimate vooludega, nagu näiteks servomootorite toitesiin ja aku toitesiin, rajad on laiemad. Et tagada trükkplaadil ühtlane ja madala impedantsiga maandus, täideti plaadil üleliigne pind, kus ei ole radasid ega jootevälju, täielikult maanduse väljaga.

Trükkplaati kasutati ka kui mehhaanika osana. Nii roboti kui ka puldi trükkplaati läbib 20 mm diameetriga auk, mis on mõeldud raadiomooduli antenni stabiliseerimiseks. Teise versiooni trükkplaadil on ka servomootorite mõõtmetele vastav laiendus tehtud plaadi külgedele.

4 *FPGA* tarkvara

FPGA konfigureerimiseks kasutati riistvarakirjelduskeelt Verilog. Koodi sünteesimiseks, implementeerimiseks ning bitijada genereerimiseks kasutati tarkvara *Altera Quartus II Web Edition* [22]. Antud paketis oli ka *Altera ModelSim*, mida kasutati Verilogi moodulite verifitseerimisel. Erinevate ajastust nõudvate probleemide, näiteks helisignaali õige perioodiga signaali genereerimine, lahendamisel lähtuti eeldusest, et FPGAl kasutatav ostsillaator genereerib 25MHz sagedusega signaali ning selle põhjal oli võimalik arvutada viiteid.

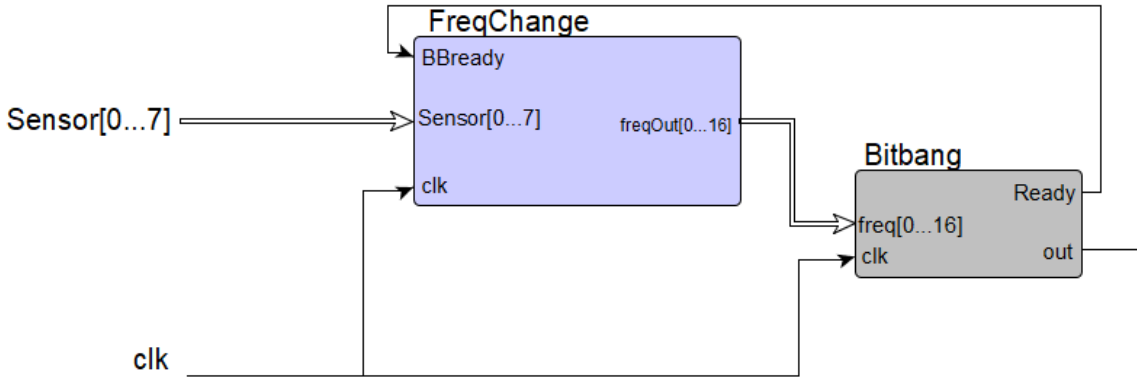
Ülemise kihi (*top-level*) moodulis on 6 alammoodulit, mille ülesanneteks on andmete kogumine, töötlemine ning väljastamine. Lisaks andmetöötlusele tehakse ka FPGAl helisignaali genereerimine. Joonis 7 kujutab tähtsamate tarkvaramoodulite omavahelisi ühendusi. Moodul *Driver* edastab moodulitele *Servo_PWM_L* ja *Servo_PWM_R* vastavate servomootorite juhtsignaali impulsi pikkuse ning need omakorda genereerivad servomootoritele vastuvõetava juhtsignaali. Moodul *FreqChange* edastab moodulile *Bitbang* sensoreitelt saadud info põhjal taasesitatavad helisagedused ning *Bitbang* genereerib selle põhjal helisignaali. Vastavate moodulite täpsemad tööpõhimõtted on kirjeldatud järgnevates alapeatükkides. Kogu programmikood on saadaval Lisas 1



Joonis 7. Lihtsustatud plokkskeem FPGA moodulitest.skeem FPGA moodulitest.

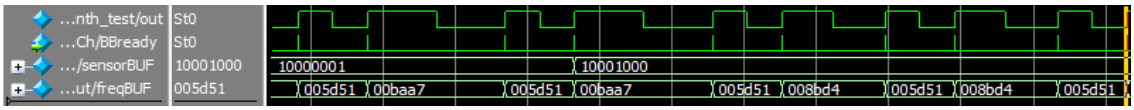
4.1 Heli genereerimine

Prototüübi arendamisel oli järgmiseks sammuks heliväljundi genereerimine. Seejuures oli oluline, et heli oleks polüfooniline. Polüfoonilist heli on võimalik genereerida üldjuhul ka erinevate sageduskomponentide liitmisel ning näiteks digitaal-analoogmuunduriga helisignaaliks muundada [23]. Kuna antud FPGAl ei ole digitaal-analoogmuundurit, siis oli võimalik kasutada ainult kahte pingetaset polüfoonilise heli genereerimisel, mis oli esialgu probleem. Lahenduseks pakuti väljundsignaali sageduste vaheldumine selliselt, et korraga taasesitati kindlal sagedusel ruutsignaal ainult ühe perioodi jooksul ning siis vahetati koheselt järgmisele sagedusele. Vaheldades signaali sagedust pärast iga signaaliperioodi edastamist, tundub inimkõrvale, et edastatakse korraga mitmesageduslikku ehk polüfoonilist heli, kuna sageduste vaheldumine toimub väga kiirelt. Minimaalseks helikõrguste vaheldumise sageduseks, mil inimkõrvale jääb mulje, et tegemist on polüfoonilise heliga, on 30 Hz [24]. Helisignaali genereerimiseks kirjutati kaks moodulit: *FreqChange*, mis tegeleb sageduste vahetamisega, ja *Bitbang*, mis tegeleb väljundpinge lülitamisega (Joonis 8).



Joonis 8. Moodulid FPGA koodis helisignaali genereerimiseks.

Joonis 9 kujutab heli väljundsignaali *out* muutumise sõltuvust valgussensorite väärtusi hoidvast registrist *sensorBUF* (kahendarv) ning hetkel genereeritavast sagedusest *freqBUF* (kuueteistkümnendarv). Lisaks on näha, kuidas sageduse muutumine toimub pärast pulssi *BBready*. Pikk pulss joonise keskel pärast valgussensorite registri väärtuste muutumist on tingitud sellest, et polüfoonilise helisignaali ($f_0 + f_7$) genereerimine polnud veel lõppenud.



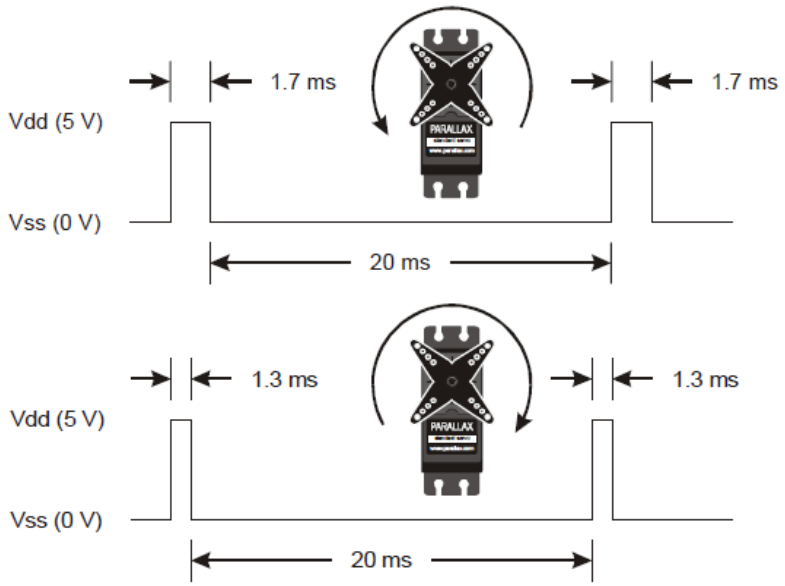
Joonis 9. Helisignaali genereerimine FPGAs.

Moodul *FreqChange* loeb väärtusi registrist, kus hoitakse väärtusi valgussensoritelt, ning annab mooduli väljundisse korraga ühele sagedusele vastava väärtuse, mis vastab taktsignaali taktide arvule, ehk mitu taktsignaali tsüklit kulub ühe poolperioodi edastamiseks. Kui on tuvastatud muutus registris, kus hoitakse väärtusi valgussensoritelt, siis „käiakse läbi“ iga sensorite registri bitt ning kui mõni neist on väärtusega „1“, siis edastatakse vastav poolperioodi pikkus. Igale sensorile vastav sageduse väärtus saab muutuda alles siis, kui moodulist *Bitbang* läheb kõrgeks vastav lubav signaal.

Moodul *Bitbang* on loodud lülitama FPGA väljaviiku, mis tüürib MOSFETi (*Metal-oxyde-semiconductor field-effect transistor*, metall-oksiid-pooljuht väljatransistor) paisu, kõrgeks ja madalaks, tekitades sellega helisignaali. Kui moodul saab sisendisse poolperioodi pikkuse taktsignaali pulsi arvudes, siis lülitatakse väljund kõrgeks ning oodatakse etteantud arv kordi taktsignaali impulsse. Seejärel lülitatakse väljund madalaks ning oodatakse ülejäänud poolperioodile vastava arvu kordi taktsignaali impulsse. Kui terve periood helisignaali on edastatud, siis antakse lubav signaal moodulile *FreqChange* uuele sagedusele vastava taktsignaalide arvu edastamiseks.

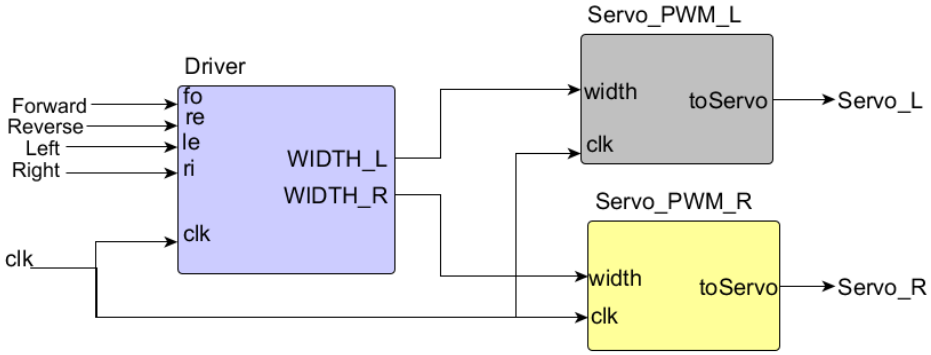
4.2 Servomootorite juhtimine

TG9e mootorid on analoog-servomootorid, mille sisendsignaaliks on pulsilaiusmoduleeritud (*PWM*) signaal perioodipikkusega 20ms. Analoogservod on digitaalse signaaliga juhitav tagasisidega alalispingemootorid, mille juhtimine käib läbi spetsiaalse servomootorite võimendi, mis on koos mootoriga ühes pakendis [25]. Mootorite kiirus ja suund sõltub signaali pulsi pikkusest: lühike pulss paneb servomootori pöörlema ühes suunas, pikk pulss teises suunas (Joonis 10).



Joonis 10. Servomootori asendi sõltuvus juhtsignaali pulsi pikkusest [26].

Teoreetiliselt peaks 1.5ms pulsi juures mootor seiskuma [8], kuid tulenevalt sellest, et ükski elektroonikakomponent pole ideaalne, on kõik mootorid unikaalsed ja vajavad eraldi kalibreerimist. Servode juhtimiseks kirjutati moodulid *Driver*, mis töötleb juhtpuldi nuppude sisendeid ning selle põhjal väljutab pulsi pikkused ja *Servo_PWM*, mis tekitab servomootoritele sobiliku juhtsignaali (Joonis 11).



Joonis 11. Servomootorite juhtsignaali genereerimine FPGAs.

Moodul *Driver* loeb registrisse kaugjuhtimispuldi nuppude väärtused. Loodi tingimused erinevate nupukombinatsioonide kohta ning sellest tulenevalt ka erinevad väljundite võimalused. Väljunditeks on mõlema mootori *PWM* signaali pulsi pikkus taktsignaali pulsi arv kordades.

Moodul *Servo_PWM* on üpris sarnane moodulile *Bitbang:* mõlemad on loodud tüürima FPGA väljaviigul pinget kõrgeks ja madalaks, kuid *Servo_PWM* moodulis arvestatakse,

et väljundsignaali perioodipikkuseks on alati 20ms (mõõdetud signaali tõusvast frondist tõusva frondini). Pulsi pikkus saadakse sisendina ning on genereeritud moodulis *Driver*. Esmalt lülitatakse väljund kõrgeks ning oodatakse eelmainitud moodulist saadud pulsi pikkuse võrra taktsignaali impulsse. Seejärel lülitatakse väljund madalaks ning oodatakse piisav arv taktsignaali, et väljundisgnaali perioodipikkus oleks 20ms.

4.3 LEDide juhtimine

Seadme trükkplaadil on veel erinevaid valgusdioode, mida saab FPGA loogikalülitustega juhtida. Nimetatud LEDe kasutati töö tarkvara arendusperioodil programmikoodi silumiseks erinevate sündmuste ning funktsioonide vahetulemuste kohta. Näiteks esmaste testimiste juures kuvati LEDidel, kas raadiomoodulist jõuavad juhtsignaalid edukalt FPGAsse või mitte.

Valgussensorite näitude kohta tagasiside saamiseks kirjutati moodul *Ledctr*. Antud moodul loendas loogiliste ühtede arvu valgussensorite andmete registris, ehk mitmesse sensorisse jõuab lävendi ületanud valgushulk. Kui selleks polnud ühtegi sensorit, lülitatakse väljund pidevalt kõrgeks. Kui ainult üks sensor oli aktiivne, siis väljund lülitatati madalaks. Olukorras, kus rohkem kui üks sensor on aktiivne, inverteeritakse väljundi väärtust iga sekundi tagant, kuna see on piisav aeg, et märgata LEDi vilkumist.

5 Parandused ning täiendused

Komplekteerides ning testides esialgset prototüüpi, ilmnesid esimesed vead ja puudujäägid elektroonikas. Peale selle ka optimeeriti ning lihtsustati skeemi ning muudeti skeemi sümboleid ühtsemaks. Lisaks kõigele muudeti ka trükkplaadi kuju, et mootorid sobiksid paremini ning sensorite vahekaugus suureneks, et vähendada joonte tuvastamist naabersensorite poolt.

5.1 Mäluseadme ja programmeerimispistiku ühendused

Esimeses versioonis üheks suurimaks probleemiks võib lugeda mäluseadme ja programmeerimispistiku vale ühendamine, mille tulemusel ei õnnestunud esialgu konfigureerida FPGAd ega ka selle välist mälu. Ühendused said tehtud antud FPGA käsiraamatu järgi, kus kirjeldati konfigureerimismeetodeid erinevate mäluseadmetega ja programmaatoritega [27]. Kuna autoril oli kasutada programmaator *USB-Blaster*, mis toetab ka kirjeldatud *Active Serial Programming*ut [28], siis otsustatigi antud protokolli kasuks. Elektriliselt ühendati mäluseade FPGAga ning programmeerimispistik samadele liinidele paralleelselt (Lisa 2).

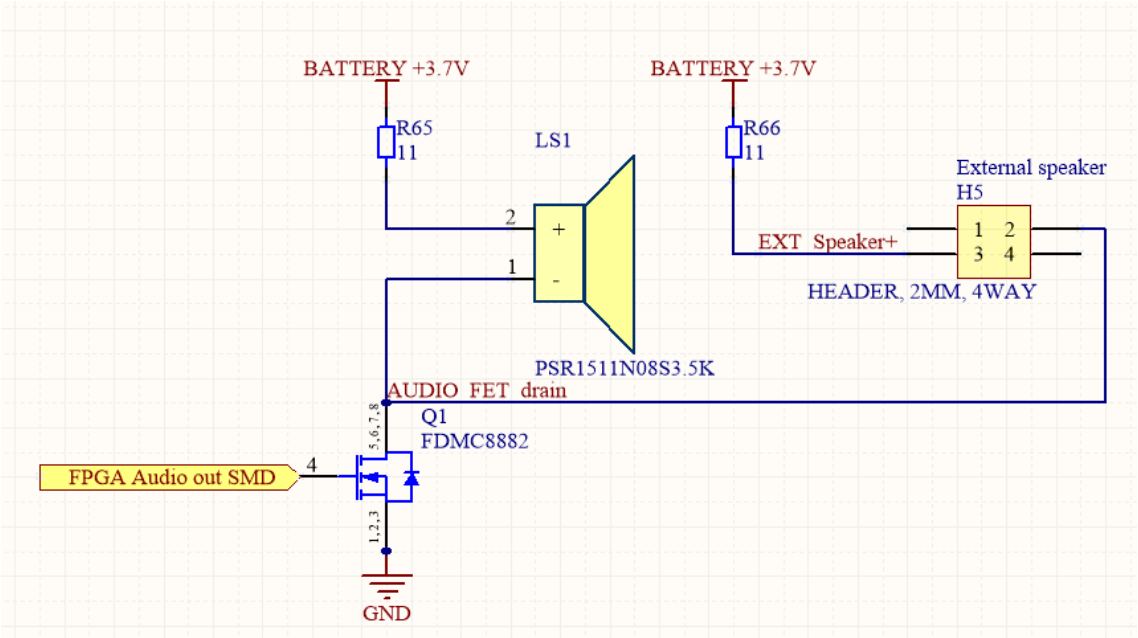
Esmakordsel FPGA konfigureerimisel ei õnnestunud antud programmaatoriga seadistada FPGAd ega ka mäluseadet. Seetõttu proovitigi konfigureerida FPGAd üle JTAG protokolli ning see õnnestus. Kuna eesmärk oli saada võimalikuks FPGA automaatne konfigureerimine mäluseadme abil igal käivitumisel, siis tuli leida ka viis selleks ning mäluseadme enda konfigureerimiseks. Selleks jäeti alles ühendused mälu ja FPGA vahel *Active Serial Programming* konfiguratsioonis. Mälu seadistamiseks tuli HDL koodist genereerida mäluseadmele sobilik bitijada, mis saadeti JTAG protokollil läbi FPGA mäluseadmesse ning pärast kogu seadme taaskäivitamist seadistati FPGA automaatselt *Active Serial Programming*u kaudu.

5.2 Joonetuvastus ning heli taasedastus

Lisaks sellele, et suurendati trükkplaadil valgussensorite vahelist kaugust, siis muudeti ning parandati ka joonetuvastuse skeemi. Algselt sooviti määrata komparaatorite inverteerivasse sisendisse tulevat etalonpinget potentsiomeetriga. Kuna erinevad pinnad peegeldavad valgust erinevalt, siis peetigi parimaks lahenduseks antud etaloni seada

potentsiomeetriga. Esmakordsel katsetamisel järeldus, et üpris raske on antud potentsiomeetreid täpse pingenivoo peale sättida ning nende üpris halva kvaliteedi tõttu asendati teises versioonis potentsiomeetrid takistitest koosneva pingejaguriga. Sellega tagati, et igale sensorile vastab sama etalonpinge.

Kuna helivõimendi skeemis oli esimeses versioonis palju puudujääke ja vigu, näiteks tagasisideahel, siis tuli ka seda muuta. Otsustati asendada kogu helivõimendi skeem N-kanali MOSFETiga, mille paisu tüüritakse FPGA helisignaali väljundiga. Neelule ühendati jadamisi kõlar otse trükkplaadile, voolu piirav takisti ning pingeallikas. Transistori lättele jäeti maapotentsiaal (Joonis 12). Lisaks jäeti ka võimalus ühendada väline kõlar, kui trükkplaadile joodetav kõlar näiteks arendusperioodil puudub.

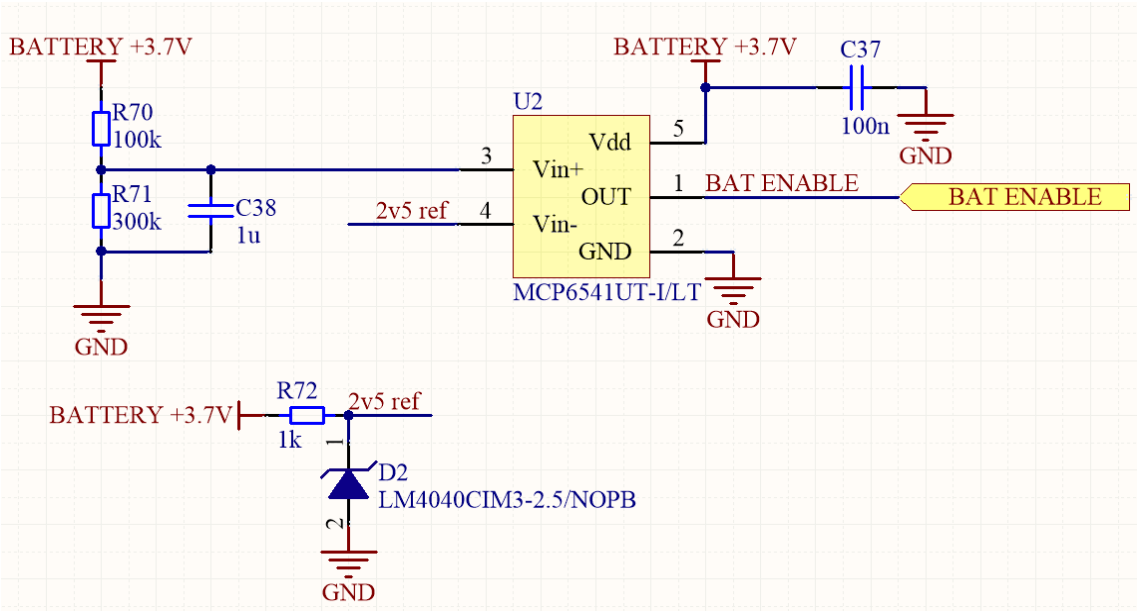


Joonis 12. Kõlari väljundkseem.

5.3 Toiteskeeem ning akukaitse

Antud seade on mõeldud töötama 3,7 V üheelemendilise LiPo (liitiumpolümeer) tüüpi aku pealt. Sellistel akudel on väga tähtis akuelemendi pingevahemik: nii ülelaadimine (ülepinge) kui ka liigne tühjakslaadimine (alapinge) on akule kahjulik ning võib tekitada lühiseid elektroodide vahel ning see omakorda võib viia põlenguteni [29]. Selleks ongi väga tähtis, et akutoitel seadmetel oleks ka alapingekaitse. Esimeses versioonis see puudus, kuid realiseeriti teises versioonis nii robotil kui ka puldil. Selleks võeti veel üks

analoogkomparaator ning inverteerivasse sisendisse ühendati stabiltronil põhinev etalonpinge 2,5 V ning mitteinverteerivasse sisendisse ühendati pingejagur akupotentsiaalist. Pingejaguri takistid R70 ja R71 valiti selliselt, et kui akupinge langeb 3,33 V, siis pingejaguril on 2,5 V ning kui pinge langeb veelgi, siis komparaatori väljund on 0V. Kui aku pinge on kõrgem kui 3,33 V, siis komparaatori väljundis on aku pinge (Joonis 13). Komparaatori väljund ühendati kõikide pingeregulaatorite lubava signaali (*enable*) sisendisse. Kuigi skeemilt on lahendus õige, siis trükkplaadil said ühendused antud komparaatori viikudele valesti tehtud, sest sümbol joonestati vale korpuse kohta.

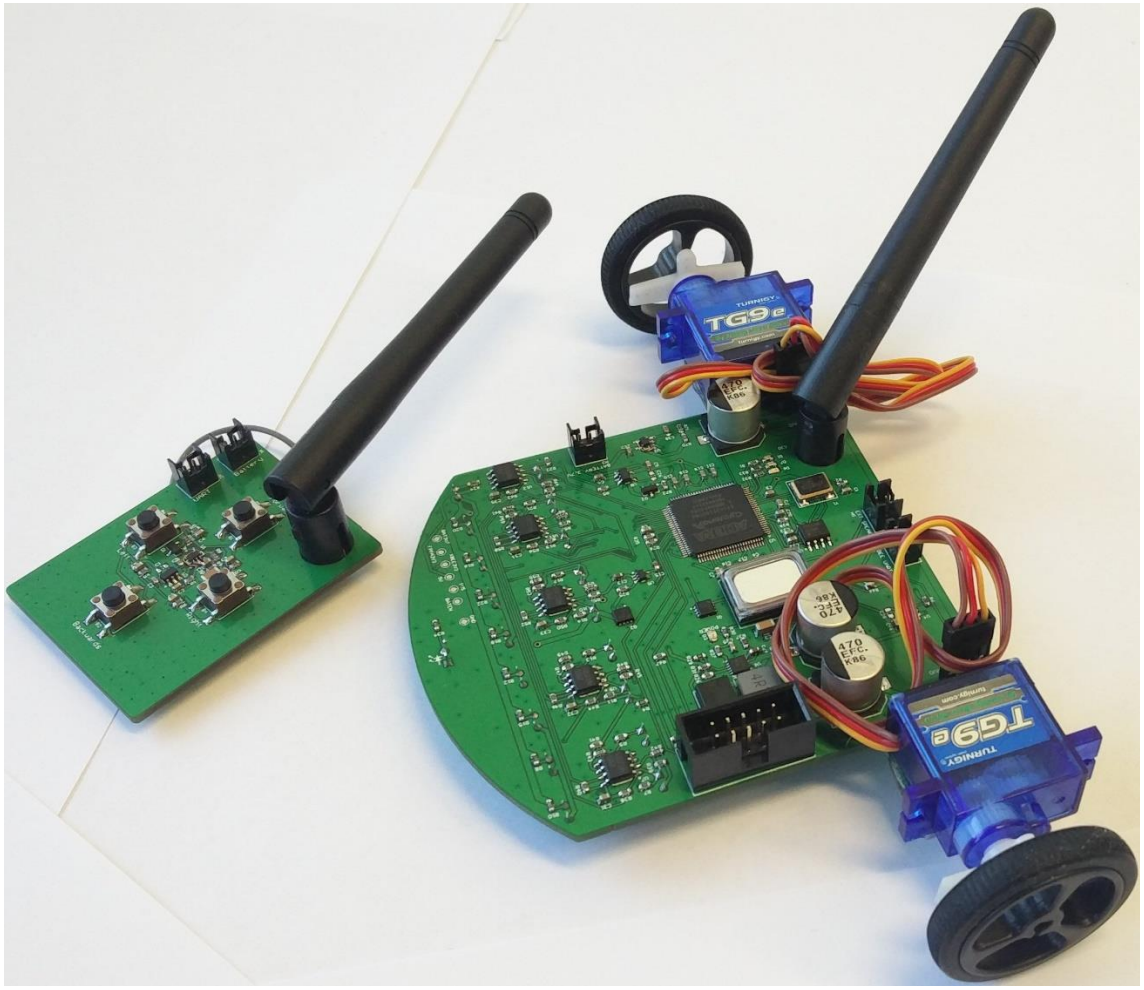


Joonis 13. Akukaitse skeem.

5.4 Lõplik seade ning edaspidised täiendused

Lõplikul prototüübil parandati kirjeldatud vead ja viidi sisse muudatused. Sellel töötavad kõik esialgses ülesandepüstituses märgitud funktsioonid, kuid ainsaks veaks on akukaitse ahela komparaatori väljaviikude funktsioonide mittevastavus trükkplaadiga, kuid need ühendused on tehtud väliselt juhtmega.

Robotile (Joonis 14) on võimalik luua veel lisaks edasiarendus ja parandada nimetaud komparaatori ahel. Lisaks võib valgussensoreid juurde lisada ning muutes FPGA koodi selliselt, et heli väljund oleks suurem kui 8-bitine. Juhtpuldile saaks luua tagasiside valgussensoritelt saadud info kohta näiteks valgusdioodide abil.



Joonis 14. Foto lõpliku roboti ja selle juhtpuldi prototüübist.

6 Kokkuvõte

Antud praktilise töö käigus disainiti trükkplaat ning komplekteeriti helisüntesaator-roboti ning kaugjuhtimispuldi trükkplaat. Seadmele tehti ka edasiarendus, mis hõlmas endas enamasti eelmise versiooni vigade parandamist, kuid lisati juurde ka uusi funktsioone nagu aku alapingekaitse. Mõlemad versioonid roboti prototüübist täitsid oma eesmärki, suutes tuvastada valgeid jooni tumedalt aluspinnalt ning genereerida antud joonte põhjal polüfoonilist heli. Lisaks oli võimalik mõlemat robotit kaugjuhtida selleks ette nähtud puldist.

Mõlemal versioonil kasutati loogikalülituste realiseerimiseks FPGA-d ning riistvarakirjelduskeelena kasutati Verilogi. Ülemise kihi (*top-level*) moodul sisaldab kuut alammoodulit, mis tegelevad andmetöötlusega. Kokku kirjutati ligikaudu 650 rida programmikoodi.

Kuna töö kummaski variandis ei kasutatud digitaal-analoogmuundureid polüfoonilise heliväljundi genereerimiseks, lahendati probleem tarkvaras. Selleks genereeriti soovitud helisagedused ruutsignaalina nii, et esitati korraga ühe helisageduse üks periood, seejärel uue sageduse periood jne. Helisageduste vaheldumine pärast iga perioodi edastamist on nii kiire, et inimene tajub seda kui mitme sageduse summana.

Seadme arendus ja disain kestis kokku ligikaudu üheksa kuud. See hõlmas endas elektroonika ja trükkplaadi disaini, kui ka tarkvara arendust. Võib väita, et töö täitis oma eesmärki, kuna koostati töötavad prototüübid ülesande püstituses kirjeldatud kaugjuhitavast polüfoonilise heli süntesaatorist.

Kasutatud kirjandus

[1] BBC, „Robots then and now,“ 22 07 2004. [Võrgumaterjal]. Saadaval: http://news.bbc.co.uk/cbbcnews/hi/find_out/guides/tech/robots/newsid_3914000/3914569.stm. [Kasutatud 22 04 2017].

[2] „The ‘Musical Telegraph’ or ‘Electro-Harmonic Telegraph’, Elisha Gray. USA, 1874,“ [Võrgumaterjal]. Saadaval: http://120years.net/the-musical-telegraphelisha-greyusa1876/. [Kasutatud 14 05 2017].

[3] Altera Corporation, „FPGA - Cyclone Series,“ Altera Corporation, [Võrgumaterjal]. Saadaval: https://www.altera.com/products/fpga/cyclone-series.html. [Kasutatud 30 10 2016].

[4] Altera Corporation, „Cyclone - Features,“ Altera Corporation, [Võrgumaterjal]. Saadaval: https://www.altera.com/products/fpga/cyclone-series/cyclone/features.html. [Kasutatud 30 10 2016].

[5] Altera Corporation, „Cyclone Handbook Volume 1, Chapter 4. DC and Switching Characteristics,“ Altera Corporation, 05 2008. [Võrgumaterjal]. Saadaval: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc/cyc_c51004.pdf. [Kasutatud 30 10 2016].

[6] Fox Electronics, „FOX MODEL: FXO-HC736R-25,“ 25 08 2008. [Võrgumaterjal]. Saadaval: http://www.farnell.com/datasheets/75459.pdf?_ga=1.56034859.1088675711.1469630034. [Kasutatud 30 10 2016].

[7] Altera Corporation, „Configuration Devices - Overview,“ [Võrgumaterjal]. Saadaval: https://www.altera.com/products/configuration-devices/overview.html. [Kasutatud 30 10 2016].

[8] D. Sawicz, „Fundamentals,“ [Võrgumaterjal]. Saadaval: http://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf. [Kasutatud 13 02 2017].

[9] HobbyKing, „Turnigy™ TG9e Eco Micro Servo 1.5kg / 0.10sec / 9g,“ [Võrgumaterjal]. Saadaval: https://hobbyking.com/en_us/turnigytm-tg9e-eco-micro-servo-1-5kg-0-10sec-9g.html. [Kasutatud 10 05 2017].

[10] Omron, „Photomicrosensor (Reflective) EE-SY171,“ [Võrgumaterjal]. Saadaval: http://www.farnell.com/datasheets/1694052.pdf?_ga=2.210825564.879072675.1494426810-1032728849.1492958756. [Kasutatud 10 05 2017].

[11] STMicroelectronics, „Low-power dual voltage comparator LM2903,“ 2013. [Võrgumaterjal]. Saadaval: http://www.st.com/content/ccc/resource/technical/document/datasheet/44/f7/9d/99/42/20/45/fd/CD00000534.pdf/files/CD00000534.pdf/jcr:content/translations/en.CD00000534.pdf. [Kasutatud 05 10 2017].

[12] Altera Corporation, „Cyclone Handbook Volume 1, Chapter 4. DC and Switching Characteristics,“ 04 2008. [Võrgumaterjal]. Saadaval: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc/cyc_c51004.pdf. [Kasutatud 05 03 2017].

[13] STMicroelectronics, „Output rail to rail 1W audio power amplifier with standby mode,“ 06 2017. [Võrgumaterjal]. Saadaval: http://www.st.com/content/ccc/resource/technical/document/datasheet/ad/2d/28/99/4c/cd/42/68/CD00002724.pdf/files/CD00002724.pdf/jcr:content/translations/en.CD00002724.pdf. [Kasutatud 10 05 2017].

[14] Mallory Sonalert Products Inc, „PSR1511N08S3.5K,“ [Võrgumaterjal]. Saadaval: http://www.mallory-sonalert.com/Specifications/PSR1511N08S3.5K.pdf. [Kasutatud 10 05 2017].

[15] HobbyKing, „Turnigy™ TG9e Eco Micro Servo 1.5kg / 0.10sec / 9g,“ [Võrgumaterjal]. Saadaval: https://hobbyking.com/en_us/turnigytm-tg9e-eco-micro-servo-1-5kg-0-10sec-9g.html. [Kasutatud 24 04 2017].

[16] Keysight Technologies, „Keysight Technologies U1230 Series Handheld Digital Multimeters (DMMs),“ 27 01 2017. [Võrgumaterjal]. Saadaval: http://literature.cdn.keysight.com/litweb/pdf/5990-7550EN.pdf?id=2043826. [Kasutatud 12 05 2017].

[17] Analog Devices, Inc, „ADP1614 (Rev. B),“ 2014. [Võrgumaterjal]. Saadaval: http://www.analog.com/media/en/technical-documentation/data-sheets/ADP1614.pdf. [Kasutatud 24 04 2017].

[18] Würth Elektronik, „Power Inductors 8 design tips,“ [Võrgumaterjal]. Saadaval: https://www.we-online.com/web/fr/index.php/show/media/07_electronic_components/download_center_1/application_notes_berichte/8_design_tipps_zur_speicherdrosselauswahl/8_Design_Tipps.pdf. [Kasutatud 06 05 2017].

[19] STMicroelectronics, „Output rail to rail 1W audio power amplifier with standby mode,“ 06 2003. [Võrgumaterjal]. Saadaval: http://www.st.com/content/ccc/resource/technical/document/datasheet/ad/2d/28/99/4c/cd/42/68/CD00002724.pdf/files/CD00002724.pdf/jcr:content/translations/en.CD00002724.pdf. [Kasutatud 05 03 2017].

[20] Texas Instruments, „MIXED SIGNAL MICROCONTROLLER,“ 04 2011. [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ds/symlink/msp430g2153.pdf. [Kasutatud 12 05 2017].

[21] Texas Instruments Incorporated, „MSP-EXP430G2 LaunchPad Evaluation Kit User's Guide (Rev. G),“ 07 2010. [Võrgumaterjal]. Saadaval: http://www.ti.com/lit/ug/slau318g/slau318g.pdf. [Kasutatud 05 03 2017].

[22] Altera Corporation, „Download Center Quartus II Web Edition,“ 06 2014. [Võrgumaterjal]. Saadaval: http://dl.altera.com/14.0/. [Kasutatud 10 05 2017].

[23] S. J. O., „Additive Synthesis (Early Sinusoidal Modeling),“ 2011. [Võrgumaterjal]. Saadaval: https://ccrma.stanford.edu/~jos/sasp/Additive_Synthesis_Early_Sinusoidal.html.

[24] P. Ungan ja S. Yagcioglu, „Significant variations in Weber fraction for changes in inter-onset interval of a click train over the range of intervals between 5 and 300 ms,“ 12 12 2014. [Võrgumaterjal]. Saadaval:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4264405/. [Kasutatud 11 05 2017].

[25] Society of Robots, „Actuators - Servos,“ [Võrgumaterjal]. Saadaval: http://www.societyofrobots.com/actuators_servos.shtml. [Kasutatud 13 02 2017].

[26] Parallax, [Võrgumaterjal]. Saadaval: http://learn.parallax.com/tutorials/robot/shield-bot/robotics-board-education-shield-arduino/chapter-2-shield-lights-servo-4. [Kasutatud 12 05 2017].

[27] Altera Corporation, „Cyclone Handbook Volume 1, Chapter 13: Configuring Cyclone FPGAs,“ 03 2008. [Võrgumaterjal]. Saadaval: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc/cyc_c51013.pdf. [Kasutatud 30 03 2017].

[28] Altera Corporation, „Intel FPGA USB Download Cable User Guide,“ 31 10 2016. [Võrgumaterjal]. Saadaval: https://www.altera.com/en_US/pdfs/literature/ug/ug_usb_blstr.pdf. [Kasutatud 30 03 2017].

[29] Woodbank Communications Ltd, „Lithium Battery Failures,“ 2005. [Võrgumaterjal]. Saadaval: http://www.mpoweruk.com/lithium_failures.htm. [Kasutatud 01 04 2017].

Lisa 1 – Programmikood

```
//Outputs audio signal if given proper input
module bitbang(
	input clk,
	input [23:0]freq, //Actually number of clk pulses per half period
	output reg out, ready
	);

	//State-machine(SM) states
	parameter START = 0, SW1 = 1, WAIT = 2, SW0 = 3, READY = 4;

	reg [23:0] counter, freqBUF;
	reg [4:0] state;

	initial begin
		out = 0;
		ready = 1;
		counter = 0;
		state = START;
		freqBUF = 0;
	end

	//SM
	always @(posedge clk) begin
		case (state)
			START : begin //Start, idle
				freqBUF = freq;
				ready = 1; // I'm ready to recieve another
frequency
				if (freq == 0) begin
					state = START;
					out = 0 ; //
				end
				else begin
					state = SW1;
				end
			end
			SW1 : begin //Switch output to 1.
				ready = 0; //I'm busy now
				out = 1;
				state = WAIT ;
			end
```

```
                    WAIT : begin //Wait for half period of current freq
                           counter = counter + 1;
                           if (counter == freqBUF) begin
                                  counter = 0;
                                  if(out) begin//Go to SW0 if out == 1, else
go READY
                                         state = SW0;
                                  end
                                  else begin
                                         //state = READY;
                                         state = START;
                                  end
                           end
                           else begin
                                  state = WAIT;
                           end
                    end
                    SW0 : begin // Switch output to 0
                           out = 0;
                           state = WAIT;
                    end

                    default : begin
                           state = START;
                    end
             endcase
      end

endmodule


//     Processes remote input signals
//     for servo rotation speeds.
//     Outputs this to pulse width modulator.
module driver (
      input clk, fo, re, ri, le,
      output reg [23:0] WIDTH_L, WIDTH_R
      );

      parameter START = 0, SW = 1 ;

      // Servo pulse lengths(in clk pulses)
      parameter STOP_R = 24'd31300, STOP_L = 24'd31295 ;
      parameter FORWARD_R = 24'd30000, FORWARD_L = 24'd32000;
      parameter FAST_FORWARD_R = 24'd25000, FAST_FORWARD_L = 24'd44500;
      parameter REVERSE_R = 24'd44500, REVERSE_L = 24'd25000;

      reg [3:0] buttons, state;

      initial begin
             buttons = 0 ;
```

```
		WIDTH_L = 0 ;
		WIDTH_R = 0 ;
		state = START ;
	end

	//	State machine
	always @(posedge clk) begin
		case  (state)

			START : begin
				//buttons = {!fo, !re, !ri, !le};	// Inverted due to xBee eval board pullups and grounding buttons,

				// remove if using custom remote!!!!
				buttons = {fo, re, ri, le};
				state = SW ;
			end

			SW : begin
				case (buttons)

					//STOP-PULSE TIME: (in my case) 1.28ms or 24'd32000 ;
					4'b1000 : begin //FORWARD
						WIDTH_L =  FAST_FORWARD_L;//24'd43750 ;	// WIDTH calculation : W=t*25000 t -  pulse time in milliseconds
						WIDTH_R =  FAST_FORWARD_R;//24'd31250 ;
					end

					4'b0100 : begin //REVERSE
						WIDTH_L = REVERSE_L ;
						WIDTH_R = REVERSE_R ;
					end

					4'b0010 : begin //RIGHT
						WIDTH_L = FAST_FORWARD_L ;
						WIDTH_R = 0 ;
					end

					4'b1010 : begin //FORWARD RIGHT
						WIDTH_L = FAST_FORWARD_L ;
						WIDTH_R = FORWARD_R ;
					end

					4'b0001 : begin //LEFT
						WIDTH_L = 0 ;
						WIDTH_R = FAST_FORWARD_R;
					end

					4'b1001 : begin //FORWARD LEFT
```

```
                                        WIDTH_L = FORWARD_L ;
                                        WIDTH_R = FAST_FORWARD_R ;
                                    end

                                    4'b0000 : begin //NEUTRAL
                                        WIDTH_L = 0 ;
                                        WIDTH_R = 0 ;
                                    end

                                    4'b1100 : begin //STOP
                                        WIDTH_L = STOP_L ;
                                        WIDTH_R = STOP_R ;
                                    end

                                    default : begin //default
                                        WIDTH_L = 0 ;
                                        WIDTH_R = 0 ;
                                    end

                                endcase

                                state = START;
                            end


                    endcase
                end


endmodule


//     Frequency changer for the bitbang module.
//     Reads values from Sensor reg and
//     outputs correspondinding frequency(number of clk pulses per half
period)
//     ONLY FOR MONOPHONIC OUTPUT, (use module below for polyphony)
module FreqChange_mono(
        input clk, BBready,
        input [7:0] sensor,
        input [2:0] activeSensors,
        output reg [23:0] freqOut
        );

        //(Here uesd to be actual frequencies, but I messed
        //     up in git and accidentally deleted this part)
        parameter freq0 = 24'd47783, freq1 = 24'd42560,
        freq2 = 24'd37925, freq3 = 24'd35796,
        freq4 = 24'd31887, freq5 = 24'd28409,
        freq6 = 24'd25309, freq7 = 24'd23889;
```

```
	reg [2:0] transmittedFreqs;
	reg  state;

	//State machine states
	parameter START = 0, SENSOR_CHECK = 1;

	initial begin
		transmittedFreqs = 0;
		freqOut = 0;
		state = START;
	end

	//State machine
	always @(posedge clk) begin
		case (state)
			START : begin
				state = SENSOR_CHECK;
			end
			SENSOR_CHECK : begin
				if(activeSensors == 3'd1) begin
					case(sensor)
						8'b00000001 : begin
							freqOut = freq0;
						end
						8'b00000010 : begin
							freqOut = freq1;
						end
						8'b00000100 : begin
							freqOut = freq2;
						end
						8'b00001000 : begin
							freqOut = freq3;
						end
						8'b00010000 : begin
							freqOut = freq4;
						end
						8'b00100000 : begin
							freqOut = freq5;
						end
						8'b01000000 : begin
							freqOut = freq6;
						end
						8'b10000000 : begin
							freqOut = freq7;
						end
						default : begin //  If more than 1
sensor is active or no sensors are active,
                                                                        //
		then output 0
							freqOut = 0;
						end
```

```
                                endcase
                                state = SENSOR_CHECK;
                        end
                        else begin
                                state = SENSOR_CHECK;
                        end
                end
        endcase
    end
endmodule

//    Frequency changer for the bitbang module.
//    Reads values from Sensor reg and
//    outputs correspondinding frequency(number of clk pulses per half
period)
module FreqChange_poly(
    input clk, BBready,
    input [7:0] sensor,
    input [3:0] activeSensors,
    output reg [23:0] freqOut
    );

    parameter freq0 = 24'd47783, freq1 = 24'd42560,
    freq2 = 24'd37925, freq3 = 24'd35796,
    freq4 = 24'd31887, freq5 = 24'd28409,
    freq6 = 24'd25309, freq7 = 24'd23889;

    reg [3:0] state, transmittedFreqs, activeSensorsBUF ;
    reg [7:0] sensorBUF ;

    parameter START = 0, SENS0 = 1, SENS1 = 2, SENS2 = 3, SENS3 = 4, SENS4
= 5, SENS5 = 6, SENS6 = 7, SENS7 = 8;

    initial begin
        freqOut = 0 ;
        transmittedFreqs = 0 ;
        activeSensorsBUF = 0 ;
        sensorBUF = 0 ;
        state = START ;
    end

    always @(posedge clk) begin
        case (state)

                        START : begin// Read values to buffers, clear
transmittedFreqs reg
                                sensorBUF = sensor ;
                                activeSensorsBUF = activeSensors ;
                                transmittedFreqs = 0 ;
                                if (activeSensors == 0) begin // if no
active sensors,
```

```
                                        state = START ;
        //      do not continue
                                        freqOut = 0 ;
                                  end
                                  else begin
                                        state = SENS0;
                                  end
                            end


                            SENS0 : begin
                                  if (sensorBUF[0]) begin  //Is sensor[0]
active?, if no go on to next sensor
                                        if(BBready) begin         // Has the
BitBang module finished

              //      outputting period of the pervious signal? If no, then
wait
                                              freqOut = freq0 ;
                                              transmittedFreqs =
transmittedFreqs + 1 ; // increment transmitterFreqs counter
                                              if (transmittedFreqs ==
activeSensorsBUF) begin   // Is every signal transmitted now?
                                                    state = START; //   if
yes, then go START
                                              end
                                              else begin
                                                    state = SENS1; //else go
on to check next sensor
                                              end
                                        end
                                        else begin
                                              state = SENS0 ;
                                        end
                                  end
                                  else begin
                                        state = SENS1 ;
                                  end
                            end

                            SENS1 : begin//For comments, look state SENS0
                                  if (sensorBUF[1]) begin
                                        if(BBready) begin
                                              freqOut = freq1 ;
                                              transmittedFreqs =
transmittedFreqs + 1 ;
                                              if (transmittedFreqs ==
activeSensorsBUF) begin
                                                    state = START;
                                              end
                                              else begin
                                                    state = SENS2;
```

```
                                        end
                                    end
                                    else begin
                                        state = SENS1 ;
                                    end
                                end
                                else begin
                                    state = SENS2 ;
                                end
                            end

                            SENS2 : begin//For comments, look state SENS0
                                if (sensorBUF[2]) begin
                                    if(BBready) begin
                                        freqOut = freq2 ;
                                        transmittedFreqs =
transmittedFreqs + 1 ;
                                        if (transmittedFreqs ==
activeSensorsBUF) begin
                                            state = START;
                                        end
                                        else begin
                                            state = SENS3;
                                        end
                                    end
                                    else begin
                                        state = SENS2 ;
                                    end
                                end
                                else begin
                                    state = SENS3 ;
                                end
                            end

                            SENS3 : begin//For comments, look state SENS0
                                if (sensorBUF[3]) begin
                                    if(BBready) begin
                                        freqOut = freq3 ;
                                        transmittedFreqs =
transmittedFreqs + 1 ;
                                        if (transmittedFreqs ==
activeSensorsBUF) begin
                                            state = START;
                                        end
                                        else begin
                                            state = SENS4;
                                        end
                                    end
                                    else begin
                                        state = SENS3 ;
                                    end
```

```
                        end
                        else begin
                                state = SENS4 ;
                        end
                end

                SENS4 : begin//For comments, look state SENS0
                        if (sensorBUF[4]) begin
                                if(BBready) begin
                                        freqOut = freq4 ;
                                        transmittedFreqs =
transmittedFreqs + 1 ;
                                        if (transmittedFreqs ==
activeSensorsBUF) begin
                                                state = START;
                                        end
                                        else begin
                                                state = SENS5;
                                        end
                                end
                                else begin
                                        state = SENS4 ;
                                end
                        end
                        else begin
                                state = SENS5 ;
                        end
                end

                SENS5 : begin//For comments, look state SENS0
                        if (sensorBUF[5]) begin
                                if(BBready) begin
                                        freqOut = freq5 ;
                                        transmittedFreqs =
transmittedFreqs + 1 ;
                                        if (transmittedFreqs ==
activeSensorsBUF) begin
                                                state = START;
                                        end
                                        else begin
                                                state = SENS6;
                                        end
                                end
                                else begin
                                        state = SENS5 ;
                                end
                        end
                        else begin
                                state = SENS6 ;
                        end
                end
```

```
                        SENS6 : begin//For comments, look state SENS0
                               if (sensorBUF[6]) begin
                                      if(BBready) begin
                                             freqOut = freq6 ;
                                             transmittedFreqs =
transmittedFreqs + 1 ;
                                             if (transmittedFreqs ==
activeSensorsBUF) begin
                                                    state = START;
                                             end
                                             else begin
                                                    state = SENS7;
                                             end
                                      end
                                      else begin
                                             state = SENS6 ;
                                      end
                               end
                               else begin
                                      state = SENS7 ;
                               end
                        end

                        SENS7 : begin//For comments, look state SENS0
                               if (sensorBUF[7]) begin
                                      if(BBready) begin
                                             freqOut = freq7 ;
                                             state = START ;     //     Next
state: START
                                      end
                                      else begin
                                             state = SENS7 ;
                                      end
                               end
                               else begin
                                      state = START ;
                               end
                        end

                        default : begin     // In case something goes wrong
and you end up here
                               state = START ;     // Go START
                        end

             endcase
      end
endmodule


//     Servo pulse width modulator.
```

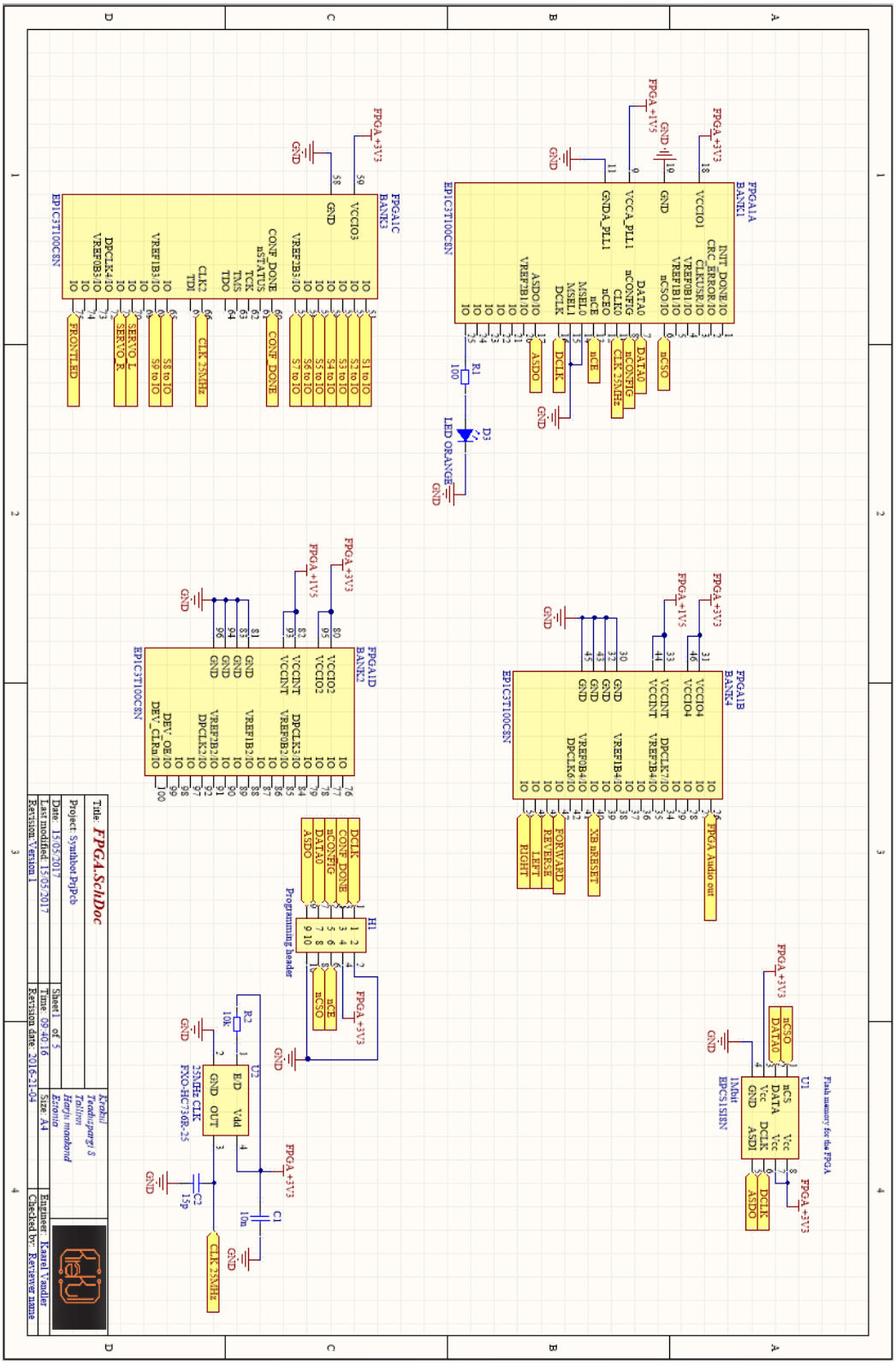
```
//     Input is pulse width in clk pulses,
//     output is 50Hz(20 ms period) servo signal
module servo_PWM(
       input clk,
       input [23:0]width,
       output reg toServo
       );

       reg [23:0] counter, widthBUF, wait0BUF;
       reg [3:0] state;
       //reg [23:0] width ;

       parameter START = 0, SW1 = 1, WAIT1 = 2, SW0 = 3, WAIT0 = 4;

       initial begin
              toServo = 0 ;
              counter = 0 ;
              state = START ;
              widthBUF = 0 ;      // Buffer to hold input pulse high(1) width
              wait0BUF = 0 ;      // Buffer to hold low (0) width
              //width = 24'd50000;
       end

       always @(posedge clk) begin
              case (state)
                                  START : begin// Read input values to buffers
                                         widthBUF = width ;
                                         wait0BUF = 23'd500000 - width ; // 500
000 clk cycles @25 MHz = 20ms
                                         if (width == 0) begin
                                                state = START ;
                                         end
                                         else begin
                                                state = SW1 ;
                                         end
                                  end

                                  SW1 : begin // Switch output to 1
                                         toServo = 1 ;
                                         state = WAIT1;
                                  end

                                  WAIT1 : begin// Wait while output is 1
                                                counter = counter + 1;
                                                if(counter == widthBUF) begin
                                                       counter = 0;
                                                       state = SW0;
                                                end
                                                else begin
                                                       state = WAIT1;
                                                end
```

```
                                end

                                SW0 : begin  //     Switch output to 0
                                       toServo = 0 ;
                                       state = WAIT0;
                                end

                                WAIT0 : begin// Wait while output is 0
                                       counter = counter + 1;
                                       if (counter == wait0BUF) begin
                                              counter = 0;
                                              state = START;
                                       end
                                       else begin
                                              state = WAIT0;
                                       end
                                end

                                default : begin
                                       state = START ;
                                end

              endcase
       end


endmodule

///gittest

// Module for giving feecback about active sensors
module ledctr(
       input clk,
       input [3:0] active_sensors,
       output reg LED
       );
       reg [23:0] counter;

       initial begin
              LED = 0;
              counter = 0;
       end

       always @(posedge clk) begin
              case (active_sensors)
                     4'd0 : begin // 0 active sensors : LED on
                            LED = 1;
                     end
                     4'd1 : begin // 1 active sensors : LED off
                            LED = 0;
                     end
```

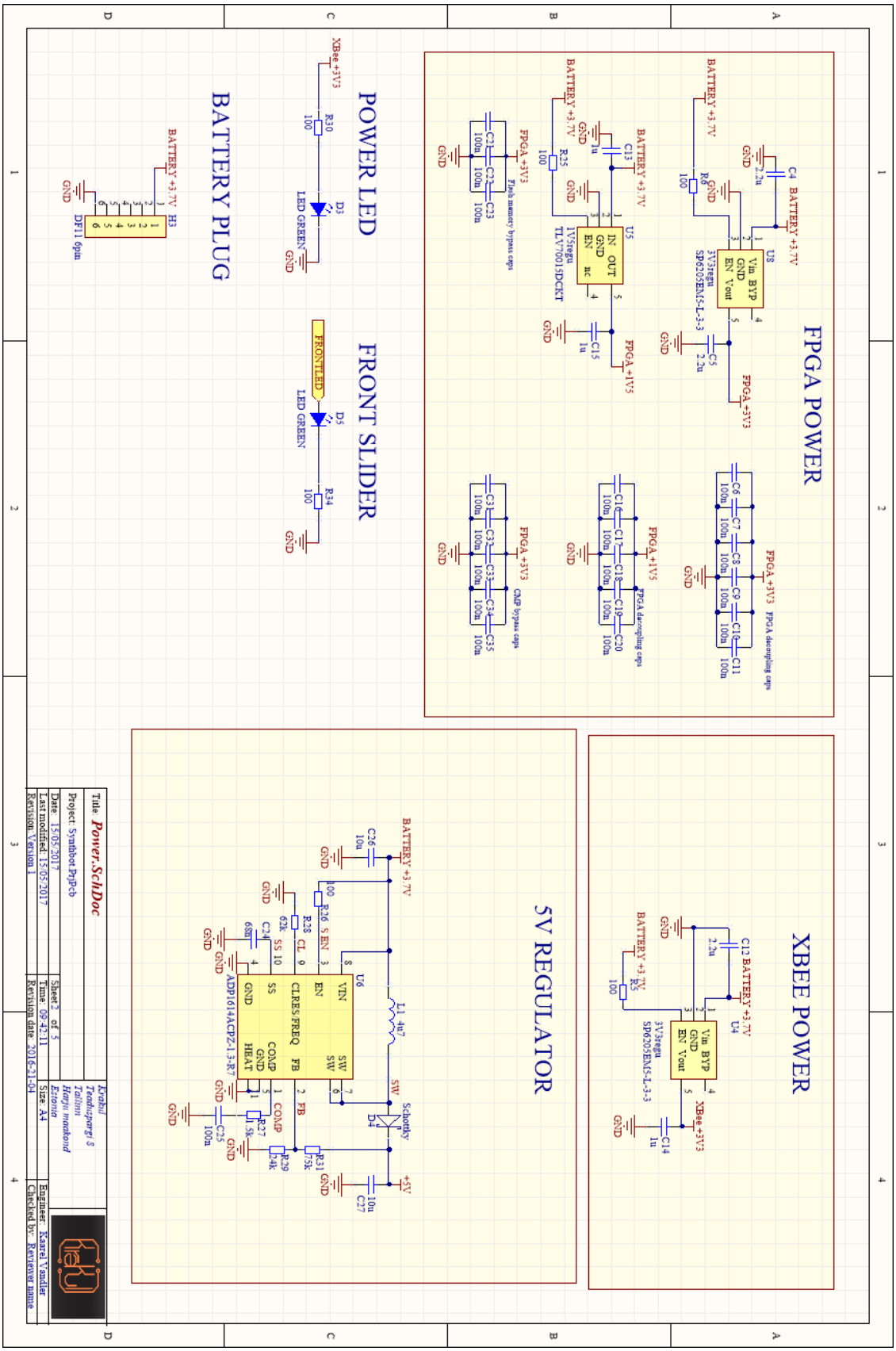
```
                default : begin // >1 active sensors : LED blinking
                        counter = counter + 1;
                        if (counter == 24'd25000000) begin
                                LED = !LED;
                                counter = 0;
                        end
                end
        endcase
    end

endmodule



//     TOP MODULE of the 8BSGB
module synth(
    input clk, Forward, Reverse, Left, Right,
    input [7:0] sensor,
    output out, smtLED, tthLED, servo_R, servo_L
    );
//
    reg [3:0] active_sensors;
    //reg [23:0] pulse;

    wire BBReady ;
    wire [23:0] pulse_L, pulse_R;
    wire [23:0] freq;


    always @(posedge clk) begin //Active sensor count
        active_sensors = sensor[0] + sensor[1] + sensor[2] + sensor[3] +
sensor[4] + sensor[5] + sensor[6] + sensor[7] ;
    end

    //for testing xBee
    assign tthLED = Forward & Reverse & Left & Right ;

    ledctr LEDmodule (
        .clk (clk),
        .active_sensors (active_sensors),
        .LED (smtLED)//smtLED
        );

    bitbang AudioOut(
        .clk (clk),
        .freq (freq),
        .out (out),
        .ready (BBReady)
        );

    FreqChange_poly freqCh(
```

```
        .clk (clk),
        .BBready (BBReady),
        .sensor (sensor),
        .activeSensors (active_sensors),
        .freqOut (freq)
        );
    /*
    servoHop servoHop_R (     //ONLY FOR DEBUGGING  servo_PWM
        .clk (clk),
        .pulseOut (pulse_R)
        );

    servoHop servoHop_L (     //ONLY FOR DEBUGGING  servo_PWM
        .clk (clk),
        .pulseOut (pulse_L)
        );
    */
    servo_PWM servo_PWM_R (
        .clk (clk),
        .width (pulse_R),
        .toServo (servo_R)
        );

    servo_PWM servo_PWM_L (
        .clk (clk),
        .width (pulse_L),
        .toServo (servo_L)
        );

    driver driver_ctr (
        .clk(clk),
        .fo(Forward),
        .re(Reverse),
        .ri(Right),
        .le (Left),
        .WIDTH_L (pulse_L),
        .WIDTH_R (pulse_R)
    );


endmodule
```

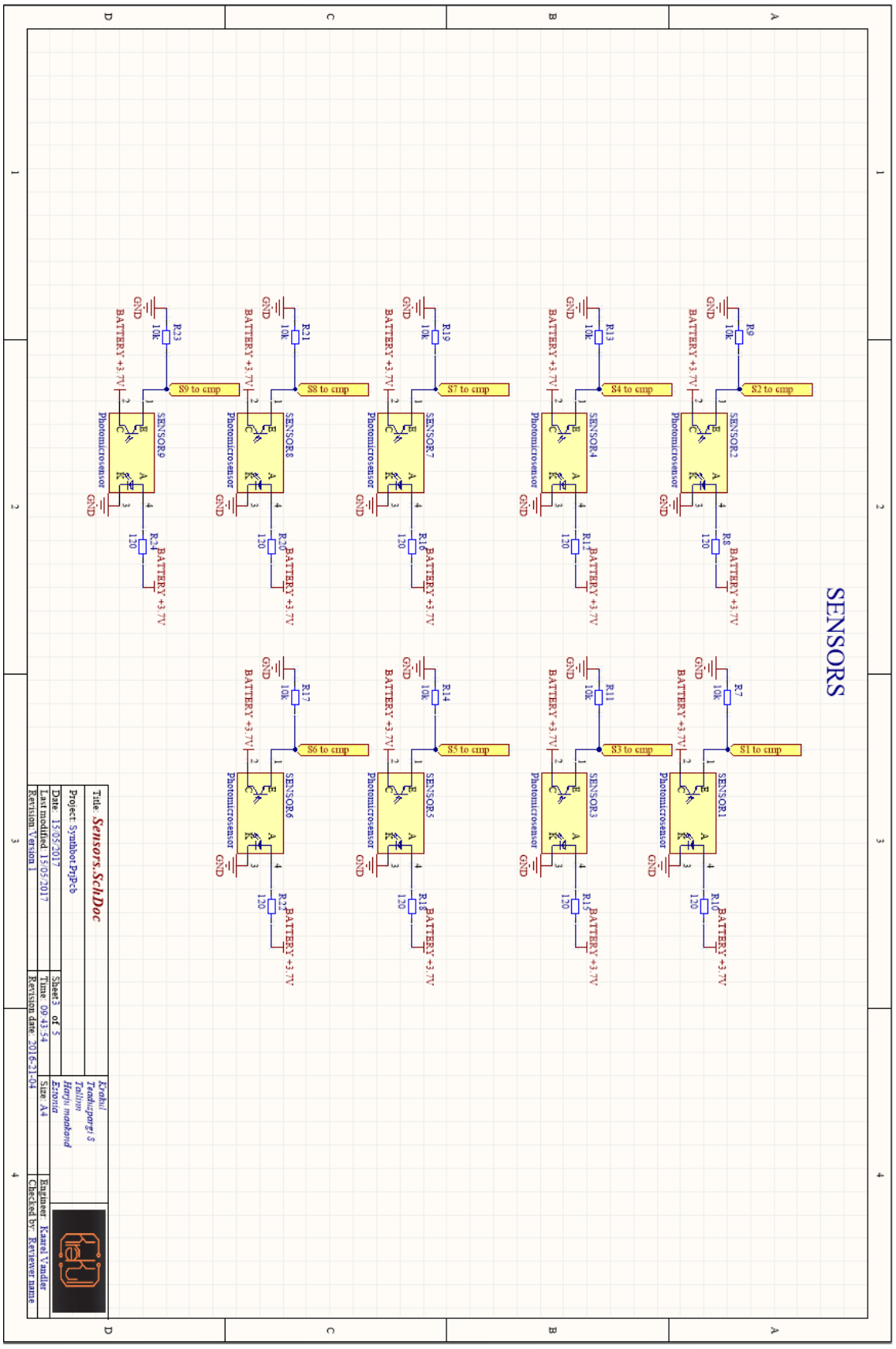
Lisa 2 – Elektriskeem ja trükkplaadid



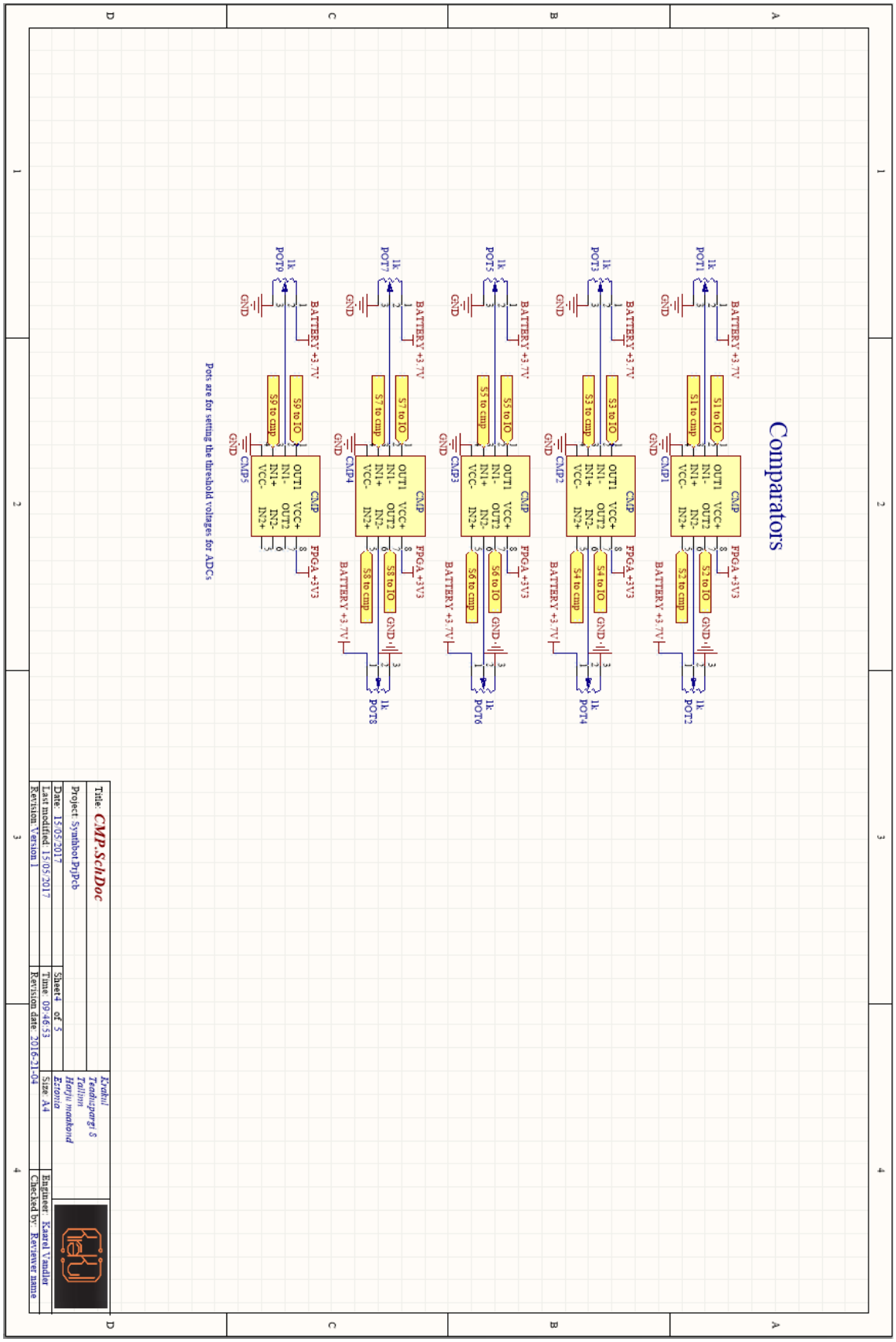
Joonis 15. Roboti elektroonikaskeemi FPGA osa versioon 1.



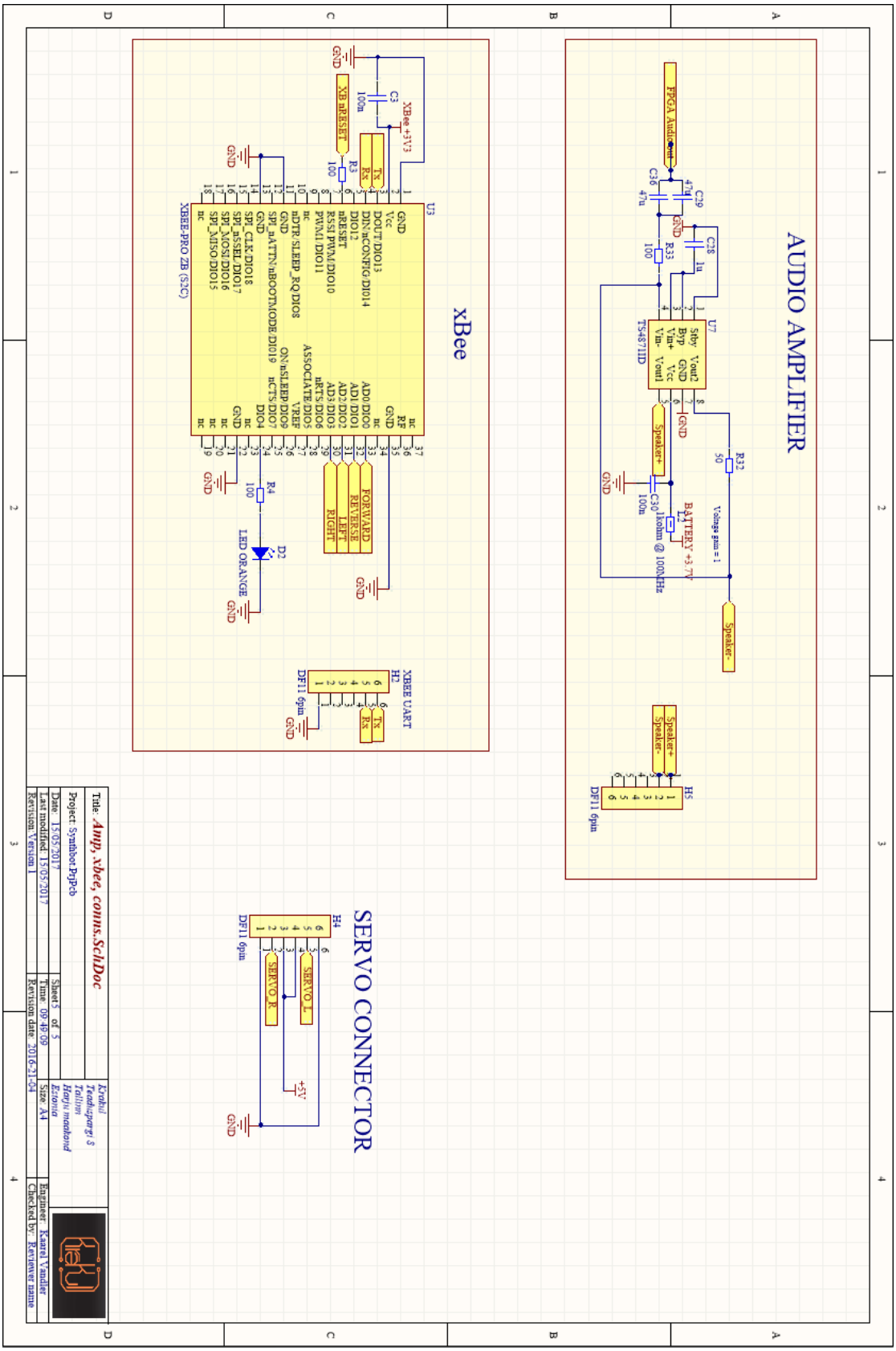
Joonis 16. Roboti elektroonikaskeemi toidete osa versioon 1.



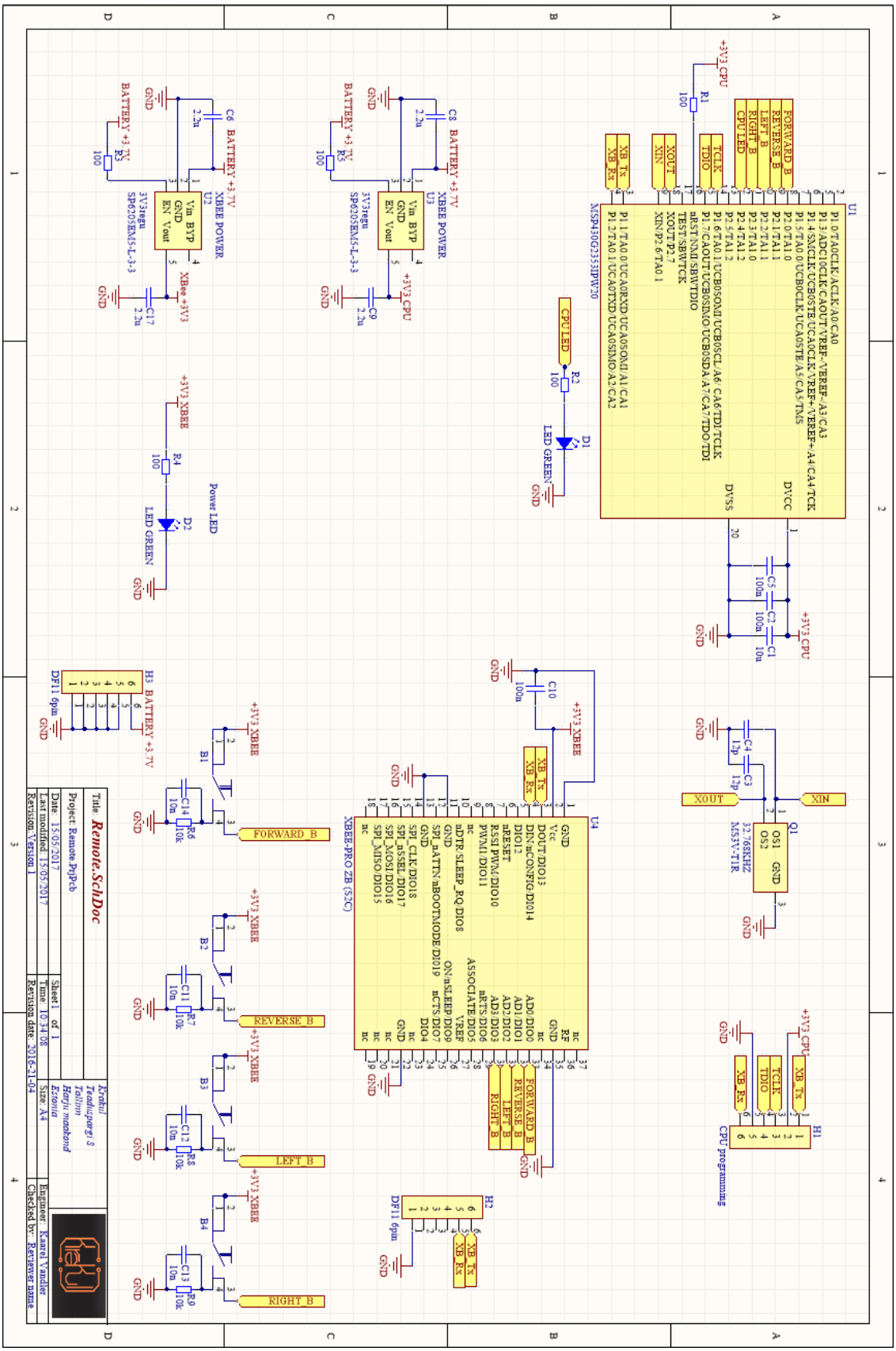
Joonis 17. Roboti elektroonikaskeemi sensorite osa versioon 1.



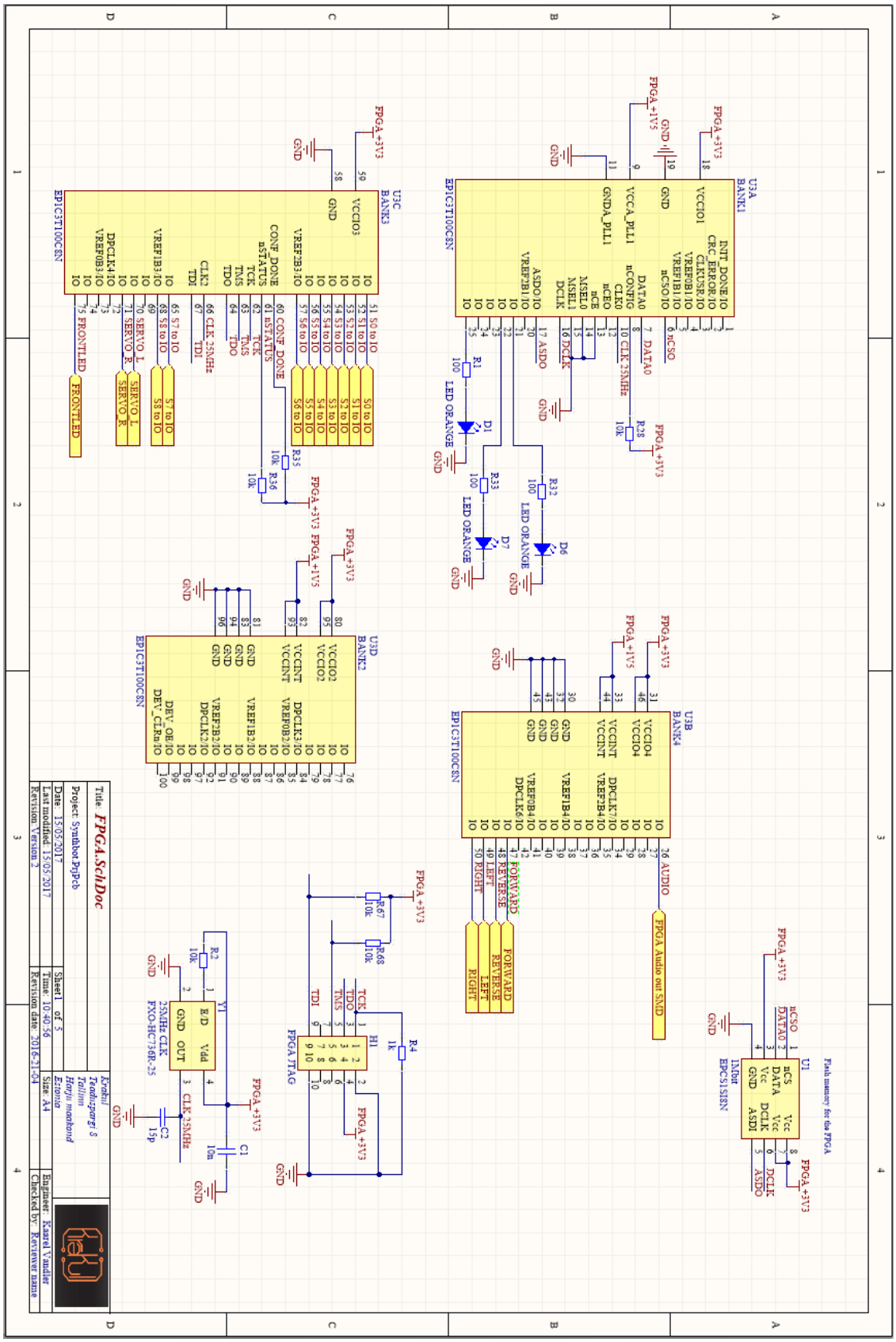
Joonis 18. Roboti elektroonikaskeemi komparaatorite osa versioon 1.



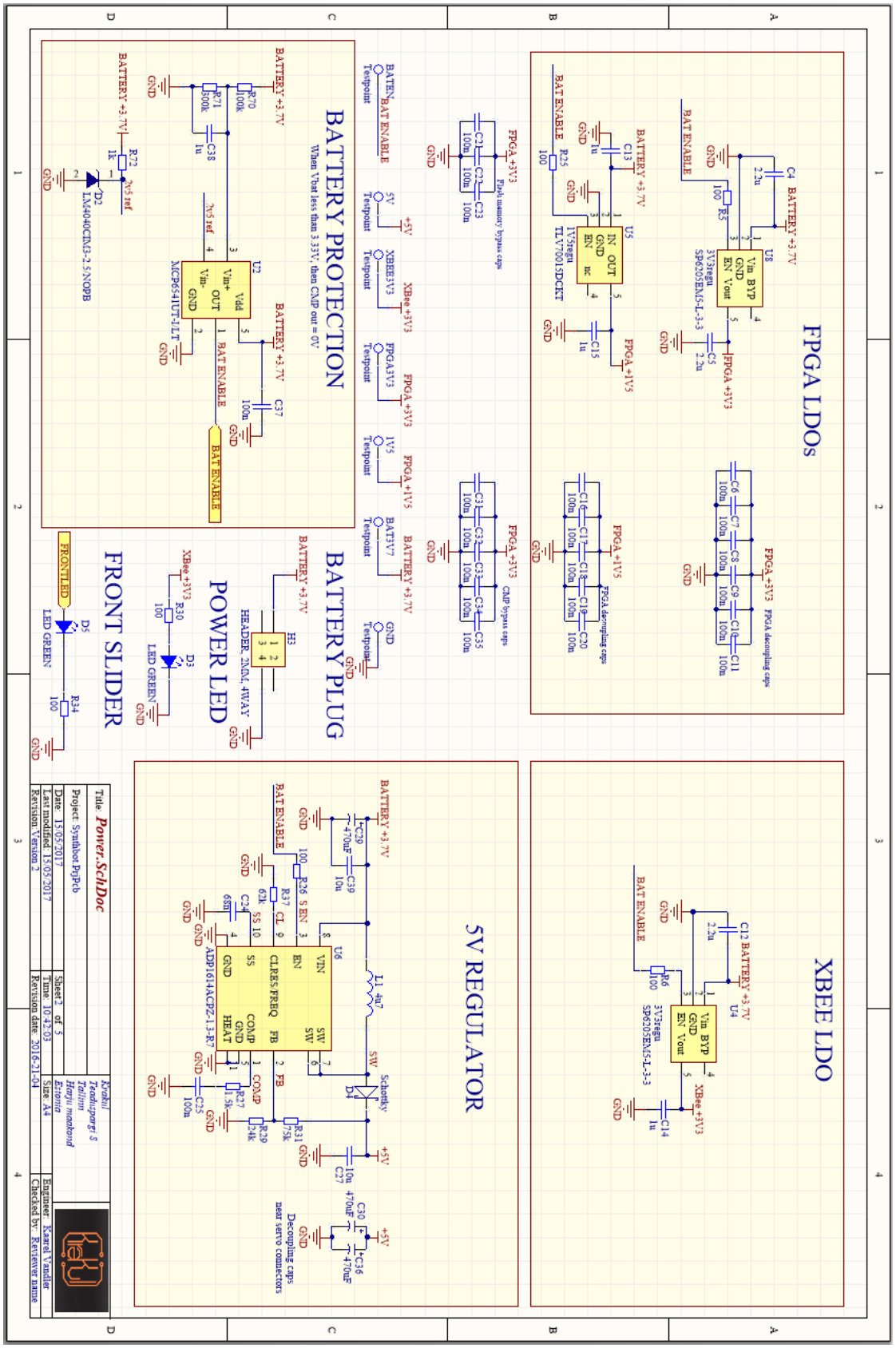
Joonis 19. Roboti elektroonikaskeemi võimendi ja xBee osa versioon 1.



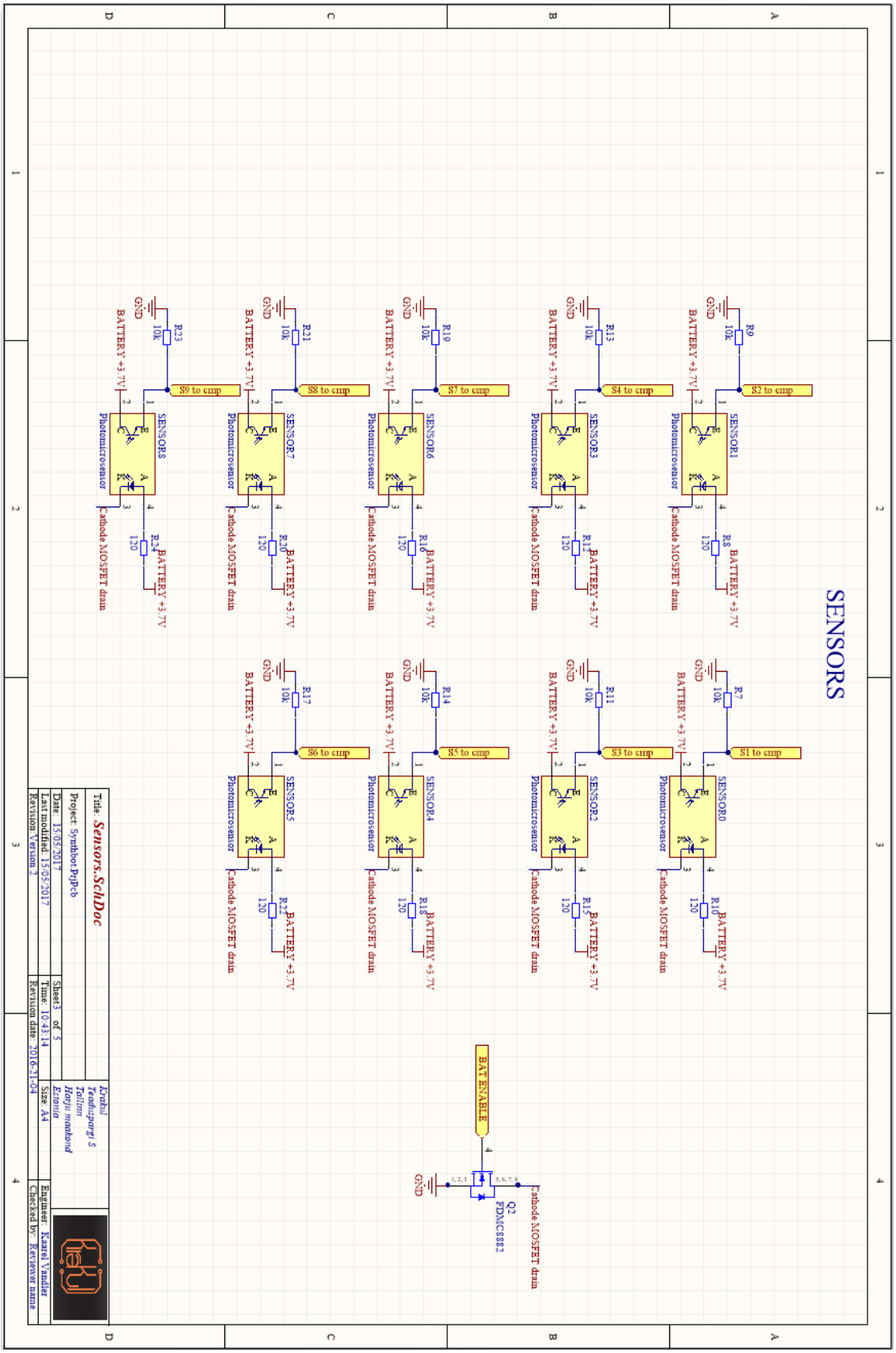
Joonis 20. Puldi elektroonikaskeem versioon 1.



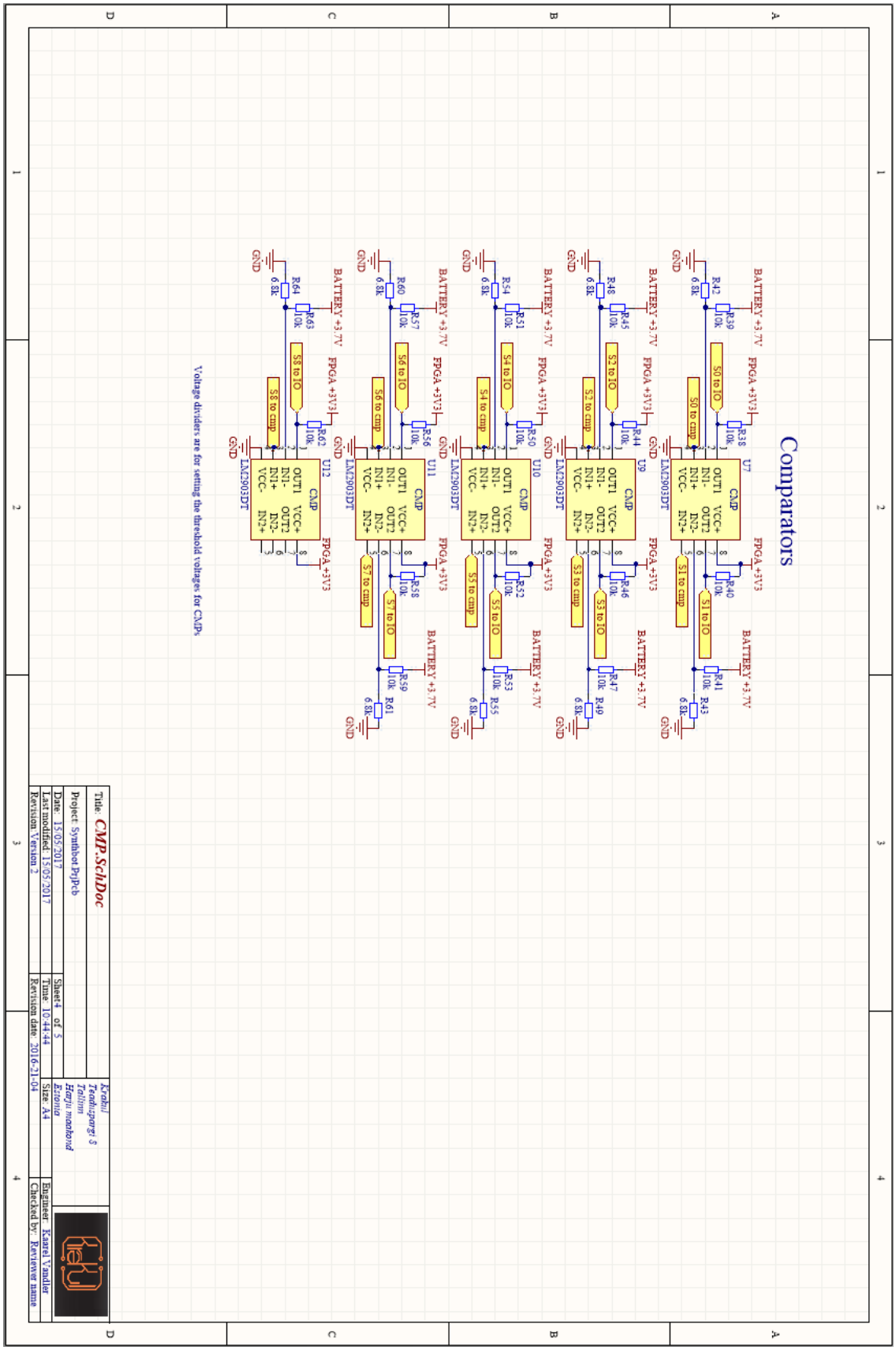
Joonis 21. Roboti elektroonikaskeemi FPGA osa versioon 2.



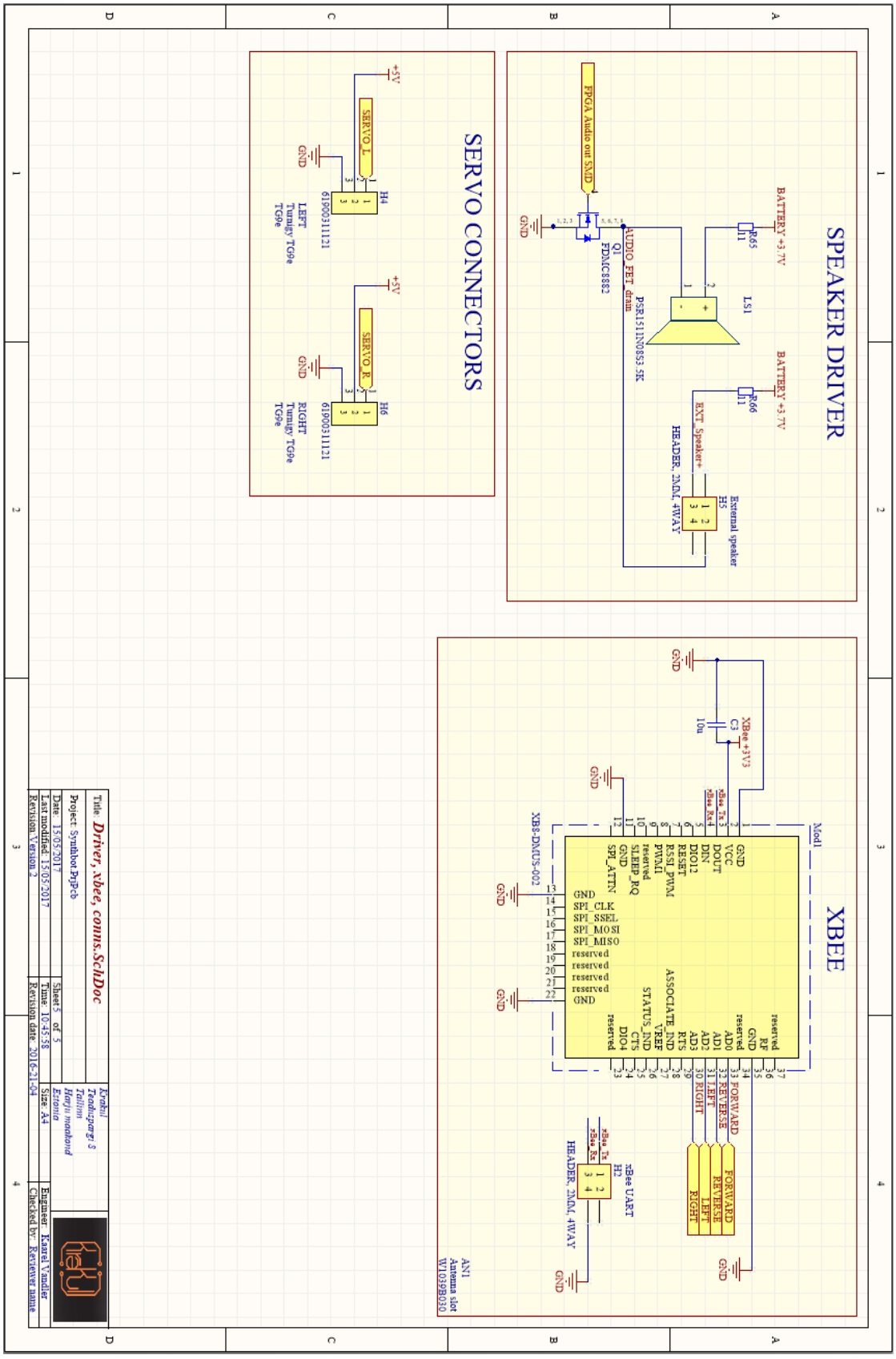
Joonis 22. Roboti elektroonikaskeemi toidete osa versioon 2.



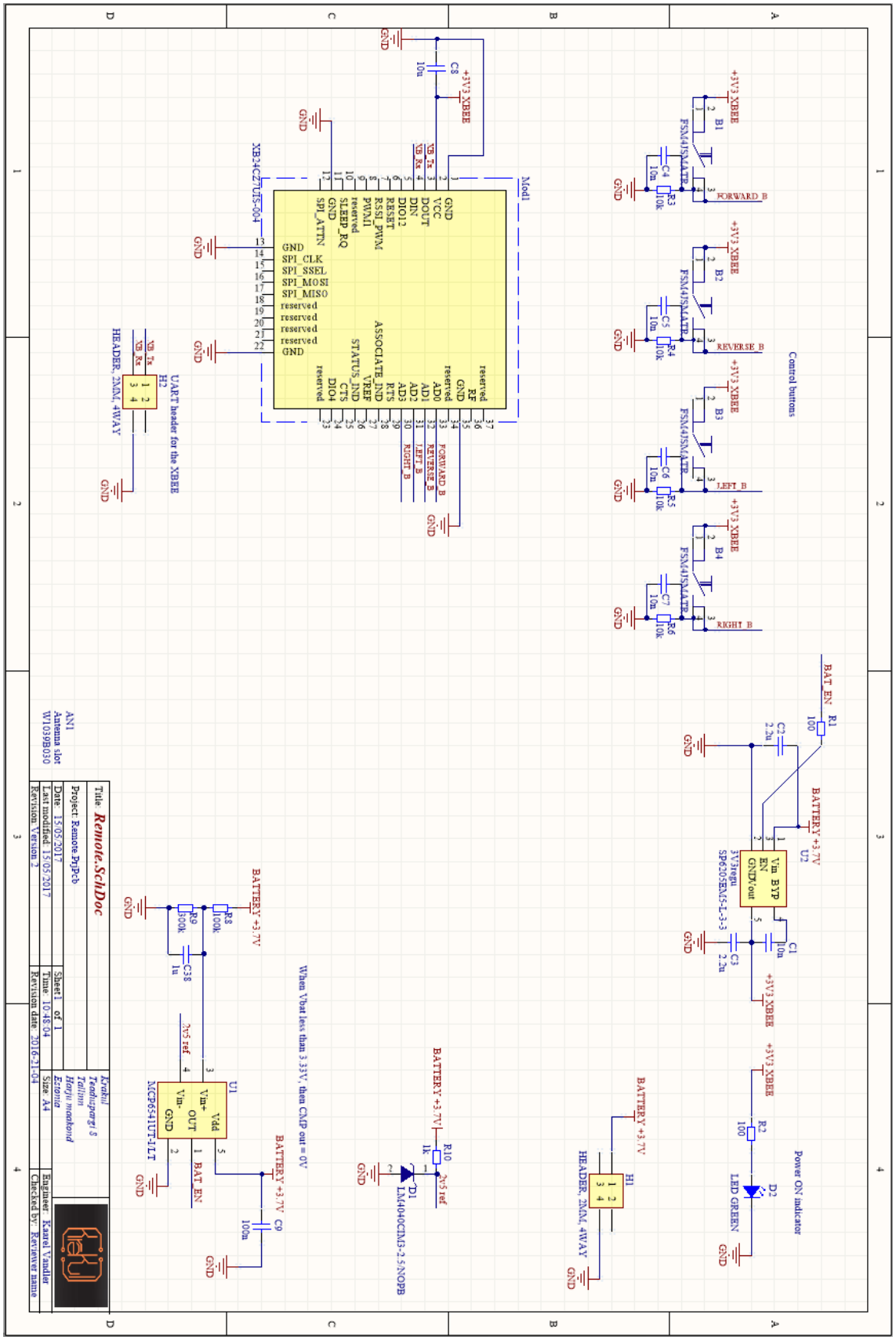
Joonis 23. Roboti elektroonikaskeemi sensorite osa versioon 2.



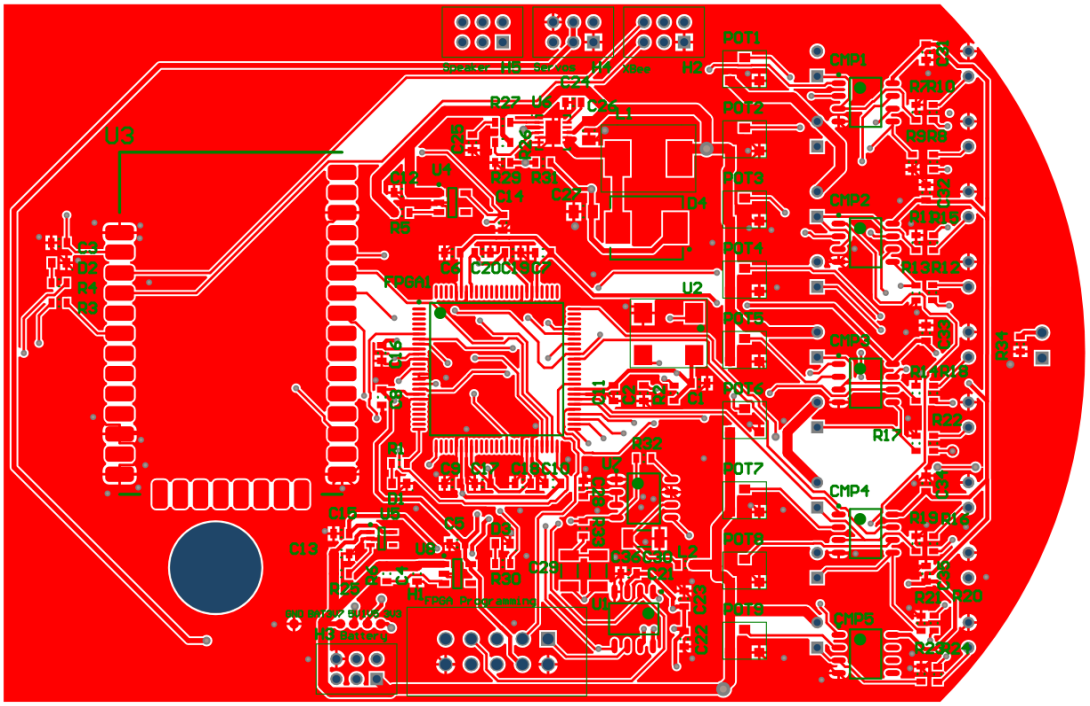
Joonis 24. Roboti elektroonikaskeemi komparaatorite osa versioon 2.



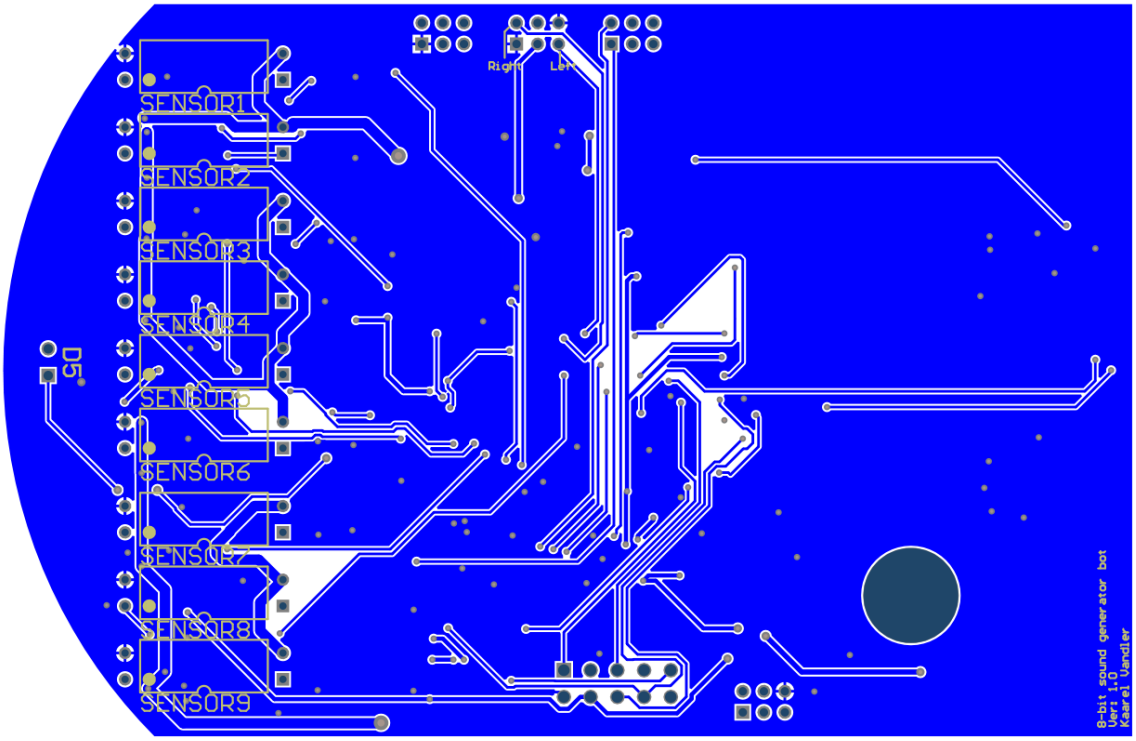
Joonis 25. Roboti elektroonikaskeemi võimendi ja xBee osa versioon 2.



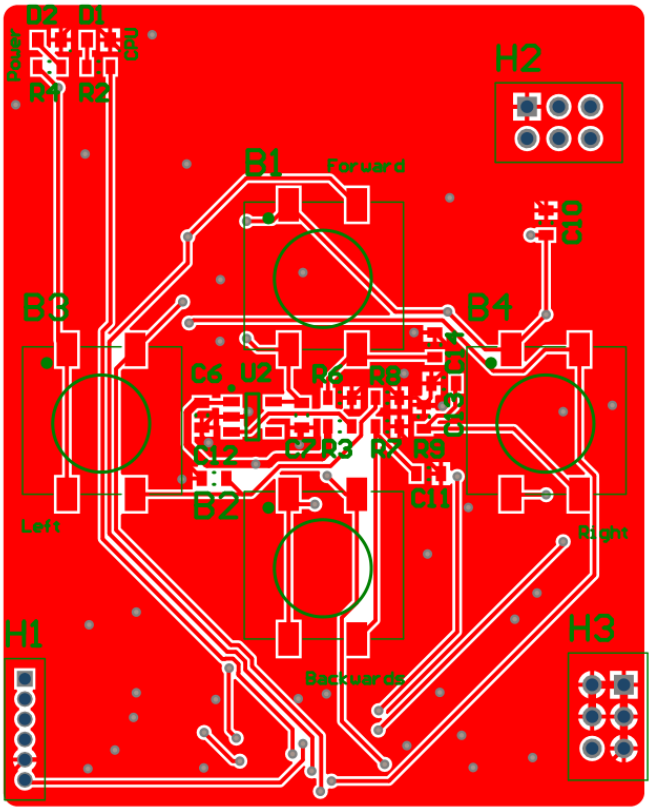
Joonis 26. Puldi elektroonikaskeem versioon 2.



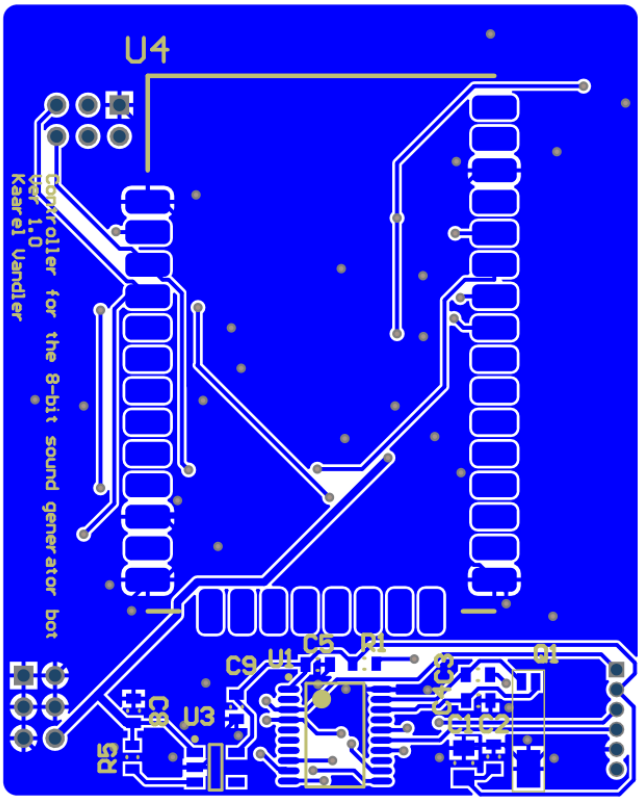
Joonis 27. Roboti esimese versiooni trükkplaadi pealmine kiht.



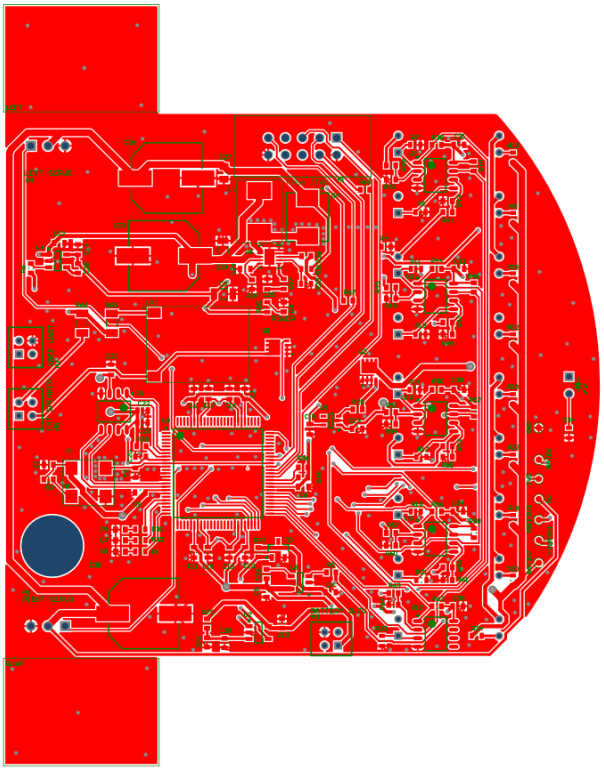
Joonis 28. Roboti esimese versiooni trükkplaadi alumine kiht.



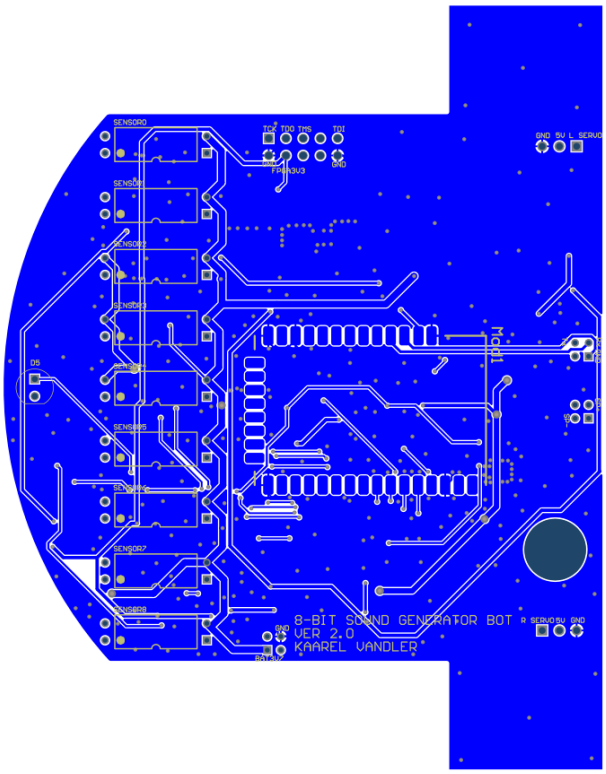
Joonis 29. Juhtpuldi esimese versiooni trükkplaadi pealmine kiht.



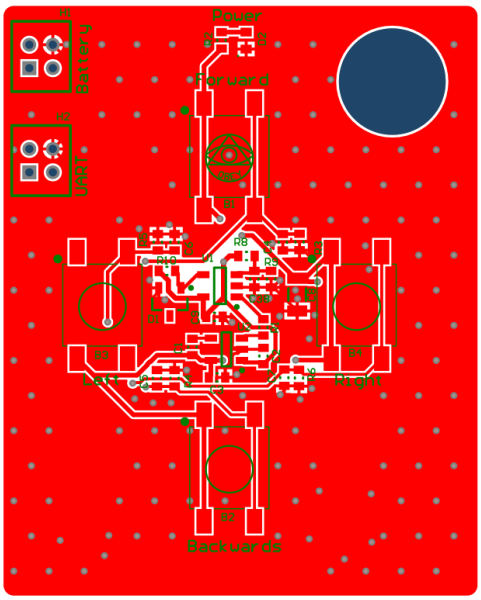
Joonis 30. Juhtpuldi esimese versiooni trükkplaadi alumine kiht.



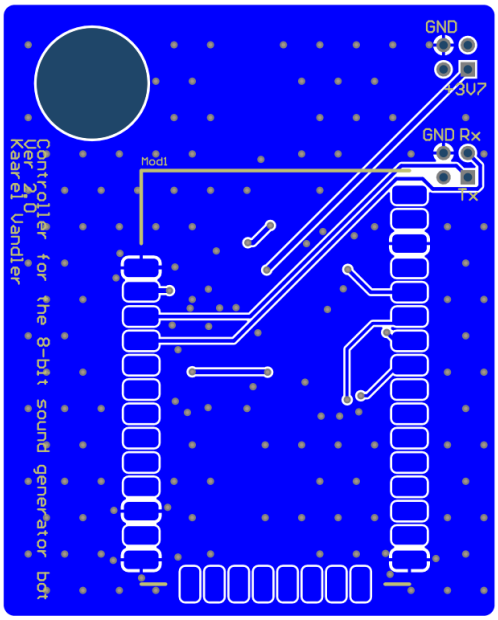
Joonis 31. Roboti teise versiooni trükkplaadi pealmine kiht.



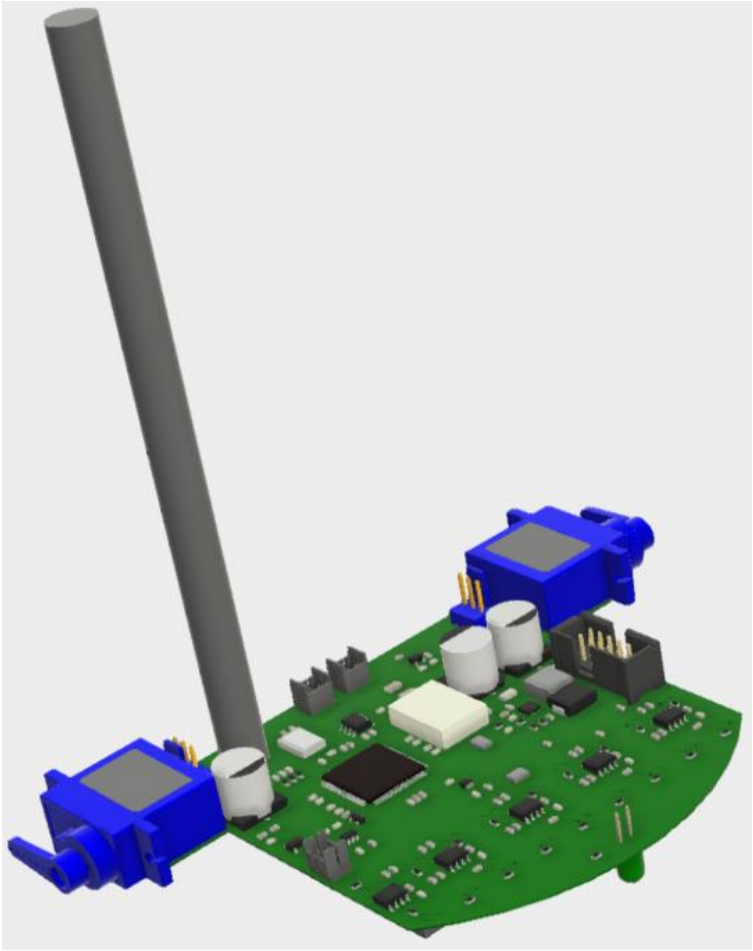
Joonis 32. Roboti teise versiooni trükkplaadi alumine kiht.



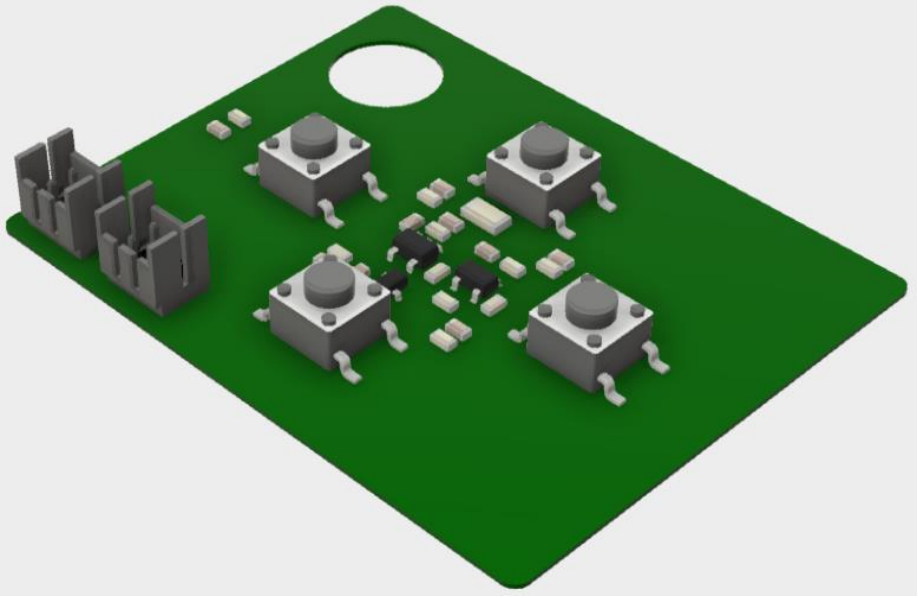
Joonis 33. Juhtpuldi teise versiooni trükkplaadi pealmine kiht.



Joonis 34. Juhtpuldi teise versiooni trükkplaadi alumine kiht.



Joonis 35. 3D-mudel lõplikust roboti komplekteeritud trükkplaadist



Joonis 36. 3D mudel lõplikust juhtpuldi komplekteeritud trükkplaadist