Latvijas Universitāte

Datorikas Fakultāte

Testa platforma Tmote Mini bezvadu sensora modulim

Kvalifikācijas darbs

Autors: Rihards Balašs

St. apl. nr.: rb09120

Vadītājs: Leo Seļāvo

Dr. dat.

Rīga 2011

Anotācija

Darbs “Testa platforma Tmote Mini bezvadu sensora modulim” apraksta gan testa platformas, gan digitālā akselerometra LIS3LV02DQ (turpmāk LIS3) dziņa izveidi. Testa platforma atbalsta ne tikai šo sensoru. Tai ir iespēja pievienot visa veida sensorus un strādāt ar tiem. Dzinis domāts, lai ar šo sensoru varētu darboties MansOS vidē, un ir rakstīts C programmēšanas valodā.

Abstract

The paper “Test module for Tmote Mini wireless sensor module” describes development of test platform and driver to digital accelerometer LIS3LV02DQ (further LIS3). Test platform supports many different sensors, which are possible to connect to test platform. The driver is designed for MansOS group, and written in C programming language.

Saturs

1 IEVADS

Tmote Mini ir kompakts "Moteiv" ražots modulis. Atšķirībā no Tmote Sky, kas vairāk ir domāts ražotājiem, kuri veido produktus, izmantojot zema enerģijas patēriņa bezvadu risinājumus, Tmote Mini ir vairāk vērsts uz darbu, kurā tiek izmantoti daudz sensori.

Šī darba mērķis ir izstrādāt daudzfunkcionālu testa platformu, kuras galvenā sastāvdaļa ir Tmote Mini.

Kvalifikācijas darba ideja radās, kad Elektronikas un Datorzinātņu institūta (EDI) zinātnieki saskārās ar problēmu, kad katram projektam, kur nepieciešams izmantot sensorus, no sākuma ir jāizveido maketa plate, uz kuras jāuzlodē nepieciešamie komponenti un sensori. Radās ideja izveidot universālu testa platformu, kurai nevajag katru reizi meklēt nepieciešamos komponentus un detaļas, lai varētu sākt darboties jaunā projektā. Šādā gadījumā ir iespējams paņemt šo testa platformu, tai klāt pievienot nepieciešamo sensoru, un sākt programmēt, testēt jaunā projekta ietvaros. Tajā pašā laikā cits projekta dalībnieks var darboties pie jaunā projekta specifiskas platformas izstrādes. Galvenais, ka programmētājam nav jāgaida pirmās testa versijas izveide, viņš jau var sākt nodarboties ar nepieciešamās programmatūras izveidi.

Lai pārbaudītu jaunās testa platformas darbību, tika nolemts izveidot dzini un testa programmu vienam bezvadu sensoram – digitālajam akselerometram LIS3LV02DQ.

Lai sasniegtu darba mērķi, nepieciešams veikt šādus uzdevumus:

- Iepazīties ar "Eagle" - mikroshēmu projektēšanas programmatūru,
- Iepazīties ar Tmote Mini un pārējiem nepieciešamajiem komponentiem,
- Iepazīties ar shēmu veidošanu pieņemto stilu,
- Aplūkot iespējamos problēmu risinājumus,
- Projektēt mikroshēmu,
- Pasūtītajai mikroshēmai atklāt kļūdas un veikt uzlabojumus,
- Kad "dzelziskajā" līmenī visas kļūdas ir novērstas, iepazīties ar digitālo akselometru LIS3LV02DQ,
- Iepazīties ar mikrokontrolieri MSP430-F1611 un I2C protokolu,
- Programmēt dzini sensoram,
- Testēt dzini ar testa programmu.

Testa platformas veidošanā liela uzmanība tiek veltīta tam, lai lietotājs varētu to ērti lietot.

Dzinis un testa programmatūra ir izstrādāta C programmēšanas valodā.

Darbs satur izstrādāto Programmatūras prasību specifikāciju, Programmatūras projektējuma aprakstu, testēšanas paraugus, reprezentatīvu dziņa un testa programmas koda paraugu, kā arī pielikumu, kurā iekļauta testa platformas shēma, tās shematiskais un reālais izskats.

2 APZĪMĒJUMU SARAKSTS

EDI – Elektronikas un Datorzinātņu Institūts

UART – *universal asynchronous receiver/transmitter tiek izmantots, lai testa platforma varētu komunicēt ar datoru caur USB*

I2C – 2 vadu saskarsne starp "Master" un "Slave"

Tmote Mini – kompakta platforma ar MSP430 mikrokontrolieri un radio

MansOS – EDI zinātnieku grupas izstrādāta operētājsistēma bezvadu sensoriem.

LIS3 – Digitālais akselerometrs LIS3LV02DQ.

LSB – least significant bit – pēdējais bits (00000001)

MSB – most significant bit – pirmais bits (10000000)

START – tiek nosūtīts start signāls

START2 – atkārtotā start procedūra

SAD+W – sensora adrese 0011101 + pēdējais bits 0 – rakstīšanas komanda

SAD+R – sensora adrese 0011101 + pēdējais bits 1 – lasīšanas komanda

SAK – *slave acknowledge* – sensora apstiprinājuma bits

RAD – reģistra adrese

MAK – *master acknowledge* – mikrokontroliera apstiprinājuma bits, lai varētu lasīt vēl vienu baitu

NMAK – *master not acknowledge* – mikrokontroliera neapstiprinājums, lai sensors pārstādu datu sūtīšanu

STOP – tiek nosūtīts stop signāls

3 PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

3.1 Ievads

3.1.1 Nolūks

Programmatūras prasību specifikācija ir paredzēta izstrādājamās testa platformas bezvadu sensora moduļa testa programmas prasību aprakstīšanai. Šis dokuments ir paredzēts programmatūras moduļa projektētājam un pasūtītājam, lai saskaņotu prasības pret produkta funkcionalitāti.

3.1.2 Darbības sfēra

Darba mērķis: izveidot daudzfunkcionālu mikroshēmu, kuras centrā ir Tmote Mini, lai nepieciešamības gadījumā, pievienojot vajadzīgo sensora moduli, var sākt darbu.

Pirmais sensors, ar kuru tiks veiktas darbības, ir digitālais akselerometrs LIS3. Šim sensoram tiks izveidots dzinis, lai ar to varētu droši strādāt iekš MansOS. Kā paraugs, lai apskatītu sensora darbību, tiks izveidota arī testa programma.

3.1.3 Saistība ar citiem dokumentiem

Programmatūras prasību specifikācija ir sastāvdaļa kvalifikācijas darbam "Testa platforma Tmote Mini bezvadu sensora modulim".

Šis dokuments ir izstrādāts saskaņā ar LVS 68:1996 "Programmatūras prasību specifikācijas ceļvedis" standartu.

3.2 Vispārējs apraksts

3.2.1 Produkta perspektīva

Izstrādājamais produkts ir dzinis digitālajam akselerometram LIS3, daudzfunkcionālai testa platformai, kas nepieciešama sensoru moduļu testēšanai.

3.2.2 Produkta funkcijas

Programmatūras produktam ir jāinicializē akselerometrs LIS3 un jāļauj Tmote Mini ar to komunicēt. LIS3 caur seriālo portu sūta visu 3 asu (x, y, z) vērtības. Tam ir jārealizē funkcijas, kas ļauj ierakstīt un nolasīt datus no sensora reģistra.

3.2.3 Lietotāja raksturiezīmes

Dzinim ir paredzētas tikai iekšējās programmēšanas saskarnes, un tās lietotāji ir programmētāji, kuriem ir pamatzināšanas par sensoru programmēšanu un vispārējo programmēšanu. Saskarne paredzēta, izmantojot MansOS izstrādes vidi un atbilstošus C header failus.

3.2.4 Pieņēmumi un atkarības

Tiek pieņemts, ka dzinis tiks lietots galvenokārt pētniecības projektos, tāpēc prasības pret laika un atmiņas sarežģītību netiek specificētas.

3.3 Konkrētās prasības

3.3.1 Funkcionālās prasības testa platformai

Testa platformai ir izvirzītas šādas funkcionālās prasības:

3.3.1.1 Plašs sensoru atbalsts. Jābūt iespējai pievienot nepieciešamo sensoru un ar to strādāt

Strādāt ir iespējams ar sensoriem, kas spēj komunicēt ar Tmote Mini, tā kā uz Tmote Mini ir MSP430 tad sensoru atbalsts ir ļoti plašs.

3.3.1.2 Vairāku barošanas avotu atbalsts. Iespējami barošanas avoti:

Darbojoties ar Tmote Mini testa platformu ir iespējams strādāt ar šādiem barošanas avotiem:

- 3V baterijas,
- Vairāk kā 3V
  - vairāk kā 3V baterijas,
  - USB 5V,
  - JTAG 5V;

3.3.1.3 Iespēja manuāli pārslēgties starp barošanas avotiem

Lai pārslēgtos starp barošanas avotiem, no sākuma ir jāsaprot vai tie ir precīzi 3V vai vārāk kā 3V (šo darbību veic ar slēdzi). Ja tiek lietoti vairāk kā 3V, ar PWR_JUMPER palīdzību var pārslēgties starp BATERIJU, USB un JTAG barošanu.

3.3.1.4 Iespēja programmēt no USB vai JTAG

Uz testa platformas, lai lietotājs varētu izvēlēties, ir uzlikti gan USB, gan JTAG konektori.

3.3.1.5 Iespēja izmērīt spriegumu ar kādu strādā platforma

Šī darbība ir iespējama, ieslēdzot VCC_SENSE slēdzi un izmērot spriegumu uz JTAG 4. pina.

3.3.2 Funkcionālās prasības sensora dzinim

3.3.2.1 Ar vienas funkcijas palīdzību nolasīt divus 8 bitu reģistrus un to vērtības saglabāt vienā 16 bitu reģistrā

Lai tiktu realizētā šī prasība, tika uzrakstīta funkcija `readRegister16`.

Ievade:

Funkcijai padod divus 8 bitu reģistrus *H un *L, visas lielās vērtības pirmā un otrā puse.

Apstrāde:

Funkcijā nolasa vienu 8 bitu reģistru un saglabā tā vērtību iekš mainīgā high.

Turpinājumā tiek nolasīta otra puse un saglabāta iekš mainīgā low. Turpmākajā darbībā tiek uzskatīts, ka high un low ir 16 bitus gari; high tiek pabīdīts 8 bitus uz kreiso pusi, savukārt low tiek ievietots high beigās.

high =

0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

low =

0	0	0	0	0	0	0	0	Y	Y	Y	Y	Y	Y	Y	Y

Kad pārbīda high par 8 bitiem pa kreisi high =

X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

result =

X	X	X	X	X	X	X	X	Y	Y	Y	Y	Y	Y	Y	Y

Izvade:

Funkcija rezultātā izvada 16 bitu lielu vērtību.

3.3.2.2 Ar vienas funkcijas palīdzību ierakstīt reģistrā tam nepieciešamo vērtību

Lai šī prasība tiktu realizēta, tika izveidota funkcija `lis3WriteRegister`.

Ievade:

Ievadē funkcijai tiek padota reģistra adrese un vērtība, ko nepieciešams ievadīt.

Apstrāde:

Funkcijā tiek nosūtīts start signāls, adrese un rakstīšanas komanda, pēc tam tiek pakāpeniski nosūtīta reģistra adrese un vērtība. Funkcijā tiek aplūkots, vai sensors sūta atpakaļ ACK (acknowledge/apstiprinājuma) bitu.

Rakstīšanas soļi:

MASTER	START	SAD+ W		RAD		DATA		STOP
SLAVE			SAK		SAK		SAK	

SAD + W – sensora adrese + rakstīšanas komanda

SAK – sensora apstiprinājuma bits

RAD – reģistra adrese

DATA – reģistra vērtība

Izvade:

Funkcija rezultātā izvada ACK bitu. Ja tas ir 0, tad ierakstīt reģistrā ir izdevies bez problēmām.

3.3.2.3 Nokonfigurēt sensoru darbam

Lai tiktu realizēta šī prasība, tika izveidota funkcija `lis3Config`.

Ievade:

Funkcijai netiek padoti nekādi mainīgie.

Apstrāde:

Funkcija ievada 1. kontroles reģistrā nepieciešamās vērtības.

Izvade:

Funkcija rezultātā izvada ACK bitu. Ja tas ir 0, tad nokonfigurēt sensoru ir izdevies bez problēmām.

3.3.3 Aparatūras ierobežojumi

Dzinis ir paredzēts EDI grupas izstrādātajai bezvadu sensoru operētājsistēmai MansOS.

3.3.4 Nefunkcionālās prasības

Programmproduktam ir izvirzītas šādas nefunkcionālās prasības:

- Dziņa kodam jākompilējas ar MansOS C kompilatoru,
- Kodam jābūt pārskatāmam, rakstītam MansOS pieņemtajā stilā.

4 PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

4.1 Ievads

4.1.1 Dokumenta nolūks

Programmatūras projektējuma apraksts (PPA) ir paredzēts, lai aprakstītu, kā izstrādājamā digitālā akselerometra LIS3 dziņa programmatūras prasību specifikācijā minētās prasības tiks realizētas projektā. Dokuments ir radīts, lai atvieglotu analīzi, plānošanu, projekta implementēšanu un lēmumu pieņemšanu projekta attīstīšanā. Tā mērķauditorija ir dziņa izstrādātāji, uzturētāji un testētāji.

4.1.2 Darbības sfēra

Izstrādājamais projekts ir neatkarīgs dzinis, kas nodrošina darbību ar digitālo akselerometru LIS3. Programmatūras projekta uzdevums ir atvieglot projektu, kas izmanto LIS3, piedāvājot gatavu sensora inicializēšanu un daļu nepieciešamo funkciju.

4.1.3 Definīcijas

Šajā dokumentā lietotie projektam specifiskie jēdzieni ir definēti darba 1. nodaļā.

4.2 Saistība ar citiem dokumentiem

Programmatūras projektējuma apraksts ir kvalifikācijas darba „Testa platforma Tmote Mini bezvadu sensora modulim” sastāvdaļa.

Šis dokuments ir lietojams kopā ar LIS3 dziņa programmatūras prasību specifikāciju.

Dokuments ir radīts vadoties pēc Latvijas Valsts standarta LVS72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai”.

4.3 Dekompozīcijas apraksts

4.3.1 Modeļu dekompozīcija

4.3.1.1 lis3AdressWrite

Tips:

Funkcija

Nolūks:

Nosūtīt sensoram start komandu, tā I2C adresi un rakstīšanas komandu.

4.3.1.2 lis3AdressRead

Tips:

Funkcija

Nolūks:

Nosūtīt sensoram start komandu, tā I2C adresi un lasīšanas komandu.

4.3.1.3 lis3Reset

Tips:

Funkcija

Nolūks:

Nosūtīt sensoram atiestates signālu, tiek iestatītas noklusētās vērtības reģistros.

4.3.1.4 lis3Config

Tips:

Funkcija

Nolūks:

Nokonfigurēt sensoru vajadzīgajam darba režīmam.

4.3.1.5 readRegister16

Tips:

Funkcija

Nolūks:

Nolasīt divus 8 bitu reģistrus un to vērtības ievadīt vienā 16 bitus garā mainīgajā.

4.3.1.6 lis3WriteRegister

Tips:

Funkcija

Nolūks:

Ierakstīt norādītajā reģistrā norādīto vērtību.

4.3.2 Datu pārraides projektējums

Šajā nodaļā tiek shematiski parādīts, kā notiek datu pārraide. Lai varētu nolasīt vairākus baitus no viena reģistra, reģistra adreses pirmais bits jānomaina uz 1.

Mikrokontrolieris ieraksta 1 baitu reģistrā

Master	START	SAD+W		RAD		DATA		STOP
Slave			SAK		SAK		SAK	

Mikrokontrolieris ieraksta vairākus baitus vienā reģistrā

Master	START	SAD+W		RAD		DATA		DATA		STOP
Slave			SAK		SAK		SAK		SAK	

Mikrokontrolieris nolasa vienu baitu no reģistra

Master	START	SAD+W		RAD		START2	SAD+R			NMAK	STOP
Slave			SAK		SAK			SAK	DATA		

Mikrokontrolieris nolasa vairākus baitus no reģistra

Master	START	SAD+W		RAD		START2	SAD+R			MAK
Slave			SAK		SAK			SAK	DATA	

Master		MAK		NMAK	STOP
Slave	DATA		DATA		

4.3.3 Detalizēts projektējums

4.3.3.1 Sensora iekšējo reģistru saraksts:

- `LIS3_OFFSET_X = 0x16,`
- `LIS3_OFFSET_Y = 0x17,`
- `LIS3_OFFSET_Z = 0x18,`
- `LIS3_GAIN_X = 0x19,`
- `LIS3_GAIN_Y = 0x1A,`
- `LIS3_GAIN_Z = 0x1B,`
- `LIS3_CTRL_REG1 = 0x20,`
- `LIS3_CTRL_REG2 = 0x21,`
- `LIS3_CTRL_REG3 = 0x22,`
- `LIS3_HP_FILTER_RESET = 0x23,`
- `LIS3_STATUS_REG = 0x27,`
- `LIS3_OUTX_L = 0x28,`
- `LIS3_OUTX_H = 0x29,`
- `LIS3_OUTY_L = 0x2A,`
- `LIS3_OUTY_H = 0x2B,`
- `LIS3_OUTZ_L = 0x2C,`
- `LIS3_OUTZ_H = 0x2D,`
- `LIS3_FF_WU_CFG = 0x30,`
- `LIS3_FF_WU_SRC = 0x31,`
- `LIS3_FF_WU_ACK = 0x32,`
- `LIS3_FF_WU_THS_L = 0x34,`
- `LIS3_FF_WU_THS_H = 0x35,`
- `LIS3_FF_WU_DURATION = 0x36,`
- `LIS3_DD_CFG = 0x38,`
- `LIS3_DD_SRC = 0x39,`
- `LIS3_DD_ACK = 0x3A,`
- `LIS3_DD_THSI_L = 0x3C,`
- `LIS3_DD_THSI_H = 0x3D,`
- `LIS3_DD_THSE_L = 0x3E,`
- `LIS3_DD_THSE_H = 0x3F,`

Adrese	**Nosaukums LIS3_***	**Skaidrojums**
0x16	OFFSET_X	Tiek ielādēts ieslēgšanas brīdī
0x17	OFFSET_Y	Tiek ielādēts ieslēgšanas brīdī
0x18	OFFSET_Z	Tiek ielādēts ieslēgšanas brīdī
0x19	GAIN_X	Tiek ielādēts ieslēgšanas brīdī
0x1A	GAIN_Y	Tiek ielādēts ieslēgšanas brīdī
0x1B	GAIN_Z	Tiek ielādēts ieslēgšanas brīdī
0x20	CTRL_REG1	Kontroles reģistrs 1
0x21	CTRL_REG2	Kontroles reģistrs 2
0x22	CTRL_REG3	Kontroles reģistrs 3

0x23	HP_FILTER_RESET	Netiek izmantots
0x27	STATUS_REG	Statusa reģistrs
0x28	OUTX_L	Izejas reģistrs X ass otrā puse
0x29	OUTX_H	Izejas reģistrs X ass pirmā puse
0x2A	OUTY_L	Izejas reģistrs Y ass otrā puse
0x2B	OUTY_H	Izejas reģistrs Y ass pirmā puse
0x2C	OUTZ_L	Izejas reģistrs Z ass otrā puse
0x2D	OUTZ_H	Izejas reģistrs Z ass pirmā puse
0x30	FF_WU_CFG	Brīvā kritiena un pārtraukumu konfigurācijas reģistrs
0x31	FF_WU_SRC	Pārtraukumu konfigurācijas turpinājums
0x32	FF_WU_ACK	Papildus reģistrs, kas atjauno FF_WU_SRC datus
0x34	FF_WU_THS_L	Glabā datus LSB
0x35	FF_WU_THS_H	Glabā datus MSB
0x36	FF_WU_DURATION	Brīvā kritiena atpazīšanas ātrums
0x38	DD_CFG	Virziena noteikšanas konfigurācijas reģistrs
0x39	DD_SRC	Virziena noteikšanas konfigurācijas turpinājums
0x3A	DD_ACK	Papildus reģistrs, kas atjauno DD_SRC datus
0x3C	DD_THSI_L	Glabā virziena datus LSB (iekšēji)
0x3D	DD_THSI_H	Glabā virziena datus MSB (iekšēji)
0x3E	DD_THSE_L	Glabā virziena datus MSB (ārēji)
0x3F	DD_THSE_H	Glabā virziena datus MSB (ārēji)

4.3.3.2 Produkta konstantes:

- `LIS3_ADDRESS_W =0x3A,`
- `LIS3_ADDRESS_R =0x3B,`
- `START = 0x87,`
- `BOOT = 0x10,`

Vērtība	**Nosaukums**	**Skaidrojums**
0x3A	LIS3_ADDRESS_W	Sensora adrese + rakstīšanas komanda
0x3B	LIS3_ADDRESS_R	Sensora adrese + lasīšanas komanda
0xE7	START	87 = 10000111 PD1/PD0/DF1/DF0/ST/Zen/Yen/Xen DF iestatīts uz 40Hz
0x10	BOOT	10 = 00010000 FS/BDU/BLE/BOOT/IEN/DRDY/SIM/DAS BOOT = 1, lai pārrakstītu noklusētās vērtības par jaunu.

4.3.4 Produkta iekšienē ir jārealizē šādas funkcijas:

```
1.  uint8_t lis3AddressWrite();
2.  uint8_t lis3AddressRead();
3.  uint8_t lis3WriteRegister(uint8_t regName, uint8_t regValue);
4.  void lis3Reset();
5.  uint8_t lis3Config();
6.  int16_t readRegister16(uint8_t regNameH, uint8_t regNameL);
```

4.3.4.1 lis3AdressWrite

Funkcija paredzēta, lai nosūtītu start signālu, I2C sensora adresi un rakstīšanas komandu.

```
1.  uint8_t lis3AddressWrite() {
2.     uint8_t error = 0;
3.     i2cStart();
4.     if((error = i2cWriteByte(LIS3_ADDRESS_W))){
5.           PRINTF("address write error\n");
6.     }
7.     return error;
8.  }
```

Rindas numurs	Skaidrojums
3	Nosūta start komandu
4	Nosūta sensora adrese, kurai pēdējais bits ir 0, tas nozīmē, ka tālāk veiksim rakstīšanas operācijas
5	Ja pēc nosūtīšanas neesam saņēmuši ACK bitu no sensora, ir kļūda
7	Nosūtam kļūdas kodu 0 – viss ir izdevies veiksmīgi

4.3.4.2 lis3AdressRead

Funkcija paredzēta, lai nosūtītu start signālu, I2C sensora adresi un lasīšanas komandu.

```
1.  uint8_t lis3AddressRead() {
2.     uint8_t error = 0;
3.     i2cStart();
4.     if((error = i2cWriteByte(LIS3_ADDRESS_R))){
5.           PRINTF("address read error\n");
6.     }
7.     return error;
8.  }
```

Rindas numurs	Skaidrojums
3	Nosūtam start komandu
4	Nosūta sensora adrese, kurai pēdējais bits ir 1, tas nozīmē, ka tālāk veiksim lasīšanas operācijas
5	Ja pēc nosūtīšanas neesam saņēmuši ACK bitu no sensora, ir kļūda
7	Nosūtam kļūdas kodu 0 – viss ir izdevies veiksmīgi

4.3.4.3 lis3WriteRegister

Funkcija paredzēta, lai ierakstītu reģistrā vērtību un nosūtītu stop signālu.

```
1.  uint8_t lis3WriteRegister(uint8_t regName, uint8_t regValue) {
2.    uint8_t error = 0;
3.    if((error = lis3AddressWrite())){
4.          return error;
5.    }else {
6.          if((error = i2cWriteByte(regName))) {
7.                return error;
8.          }else {
9.                if((error = i2cWriteByte(regValue))) {
10.                     return error;
11.               }
12.         }
13.   }
14.   i2cStop();
15.   return error;
16. }
```

Rindas numurs	Skaidrojums
3	Nosūtam sensoras adresi + rakstīšanas komandu
4	Ja ir kļūda ejam ārā no funkcijas (darbība atkārtojas 7. Un 10. Solī)
6	Ja sensors ir pieņēmis komandu nosūtam reģistra nosaukumu
9	Ja sensors ir pieņēmis komandu nosūtam vērtību, ko ierakstīt reģistrā
14	Nosūtam stop komandu
15	Nosūtam kļūdas kodu 0 – viss ir izdevies veiksmīgi

4.3.4.4 lis3Reset

```
1.  void lis3Reset() {
2.    if(lis3WriteRegister(LIS3_CTRL_REG2, BOOT)) {
3.          PRINTF("BOOT FAILED\n");
4.    }
5.    mdelay(1);
6.  }
```

Rindas numurs	Skaidrojums
2	Ierakstam 2. Kontroles reģistrā BOOT bitu, lai ielādētu noklusētās reģistru vērtības
3	Neveiksmes gadījumā paziņojam, ka atiestate nav izdevusies
5	Nogaidam 1ms

4.3.4.5 lis3Config

Funkcija paredzēta, lai pamodinātu sensoru no gulēšanas režīma, un nokonfigurētu sensora reģistrus vajadzīgajā režīmā.

```
1.  uint8_t lis3Config() {
2.    uint8_t error = 0;
3.    lis3Reset();
4.    if((error = lis3WriteRegister(LIS3_CTRL_REG1, START))) {
5.          PRINTF("START failed");
6.    }
7.    return error;
8.  }
```

Rindas numurs	Skaidrojums
3	Ierakstam 1. Kontroles reģistrā nepieciešamās vērtības
4	Kļūdas gadījumā ejam ārā no funkcijas
8	Nosūtam kļūdas kodu 0 – viss ir izdevies veiksmīgi

4.3.4.6 readRegister16

```
1.  int16_t readRegister16(uint8_t regNameH, uint8_t regNameL) {
2.    int8_t high=0;
3.    uint8_t low=0, error = 0;
4.    int16_t result=0;
5.    if((error = lis3AddressWrite())) {
6.          return error;
7.    }else {
8.          if((error = i2cWriteByte(regNameH))) {
9.                return error;
10.         }else {
11.               if((error = lis3AddressRead())) {
12.                     return error;
13.               }else {
14.                     high = i2cReadByte(I2C_NO_ACK);
15.                     i2cStop();
16.               }
17.         }
18.   }
19.   mdelay(100);
20.   if(error) {
21.         PRINTF("first byte read error\n");
22.         return error;
23.   }
24.   if((error = lis3AddressWrite())) {
25.         return error;
26.   }else {
27.         if((error = i2cWriteByte(regNameL))) {
28.               return error;
29.         }else {
30.               if((error = lis3AddressRead())) {
```

```
31.                     return error;
32.               }else {
33.                     low = i2cReadByte(I2C_NO_ACK);
34.                     i2cStop();
35.               }
36.         }
37.    }
38.    if(error) {
39.          PRINTF("second byte read error\n");
40.    }
41.    result = (((int16_t)(high<<8)) + (uint16_t)low);
42.    return result;
43. }
```

Rindas numurs	Skaidrojums
5	Nosūtām sensora adresi ar rakstīšanas komandu
6	Kļūdas gadījumā ejam ārā no funkcijas (darbība atkārtojas 9., 12., 22., 25., 28. Un 31 solī)
8	Nosūtām 1. reģistra adresi
11	Atkārtoti nosūtām starta darbības, tikai rakstīšanas komandas vietā nosūtam lasīšanas komandu
14	Nolasām vēlamā reģistra vērtību un saglabājam to high mainīgajā
15	Nosūtām stop signālu
20	Ja ir radusies kļūda, izdodam paziņojumu un ejam ārā no funkcijas
33	Nolasām vēlamā reģistra vērtību un saglabājam to low mainīgajā
41	Ja nav radusies neviena kļūda, saglabājam rezultātu, abus (high un low) mainīgos uztverot kā 16 bitu garus, saskaitām kopā
42	Nosūtām kļūdas kodu 0 – viss ir izdevies veiksmīgi

4.4 Testēšanas dokumentācija

Programmatūras produkta testēšanai tika veiktas šādas darbības:

- Oscilogrāfā tika pārbaudīti ienākošie un izejošie signāli,
- Uzrakstīta testa programma, kas izmanto dzinī esošās funkcijas.

4.4.1 Uzrakstīta testa programma, kas izmanto dzinī esošās funkcijas

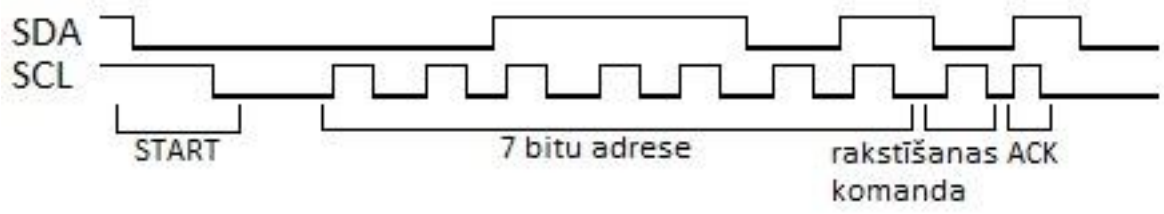
Uzrakstītās funkcijas kods:

```
#include "stdmansos.h"
#include "gpio.h"
#include "dprint.h"
#include "i2c_soft.h"
#include <hil/lis3lv02dq/lis3.h>

void appMain(void)
{
	PRINT_INIT(127); //inicialise print function
	uint8_t drdy=0, error = 0;
	int16_t x=0, y=0, z=0;
	i2cInit();
	error = lis3Config(); //configuration function
	if(error > 0) { //if error don't continue
		PRINTF("ERROR!!!\n");
		while(1) {
			toggleRedLed();
			mdelay(100);
		}
	}
while(1){ //aftter sensor configuration starts data reading
	x = readRegister16(LIS3_OUTX_H, LIS3_OUTX_L); //x
	y = readRegister16(LIS3_OUTY_H, LIS3_OUTY_L); //y
	z = readRegister16(LIS3_OUTZ_H, LIS3_OUTZ_L); //z
	//print axis values
	PRINTF(" X=%d ", x);
	PRINTF(" Y=%d ", y);
	PRINTF(" Z=%d\n", z);
	mdelay(100); //delay 100ms
}}
```

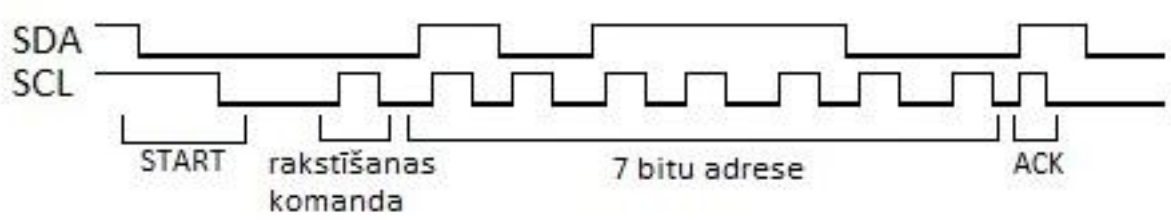
Testēšanas rezultātā radās kļūda – sensors nesūtīja atpakaļ ACK bitu. Lai patiešām pārliecinātos, ka no mikrokontroliera sūtās dati, testa platforma tika pievienota pie oscilogrāfa.

Oscilogrāfā tika ieraudzīta šāda bilde:



Pēc šiem datiem tika secināts, ka START signāls izpildās precīzi, arī 7 bitu adrese 0011101 un rakstīšanas komanda 0 tiek nosūtīti pareizi, bet no sensora netiek saņemts ACK bits, pie ACK takts signāla SDA signālam vajadzētu būt 0.

Tika nolemts aizsūtīt inverso adresi:

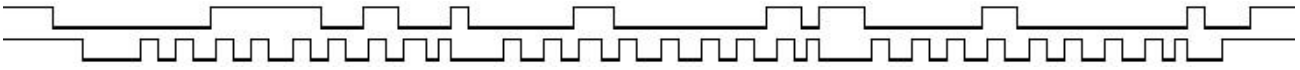


Kā redzams, START komanda atkal izpildās precīzi un arī 7 bitu adrese 1011100 ar rakstīšanas komandu 0 nosūtās precīzi. Atkal netiek nosūtīts ACK bits no sensora.

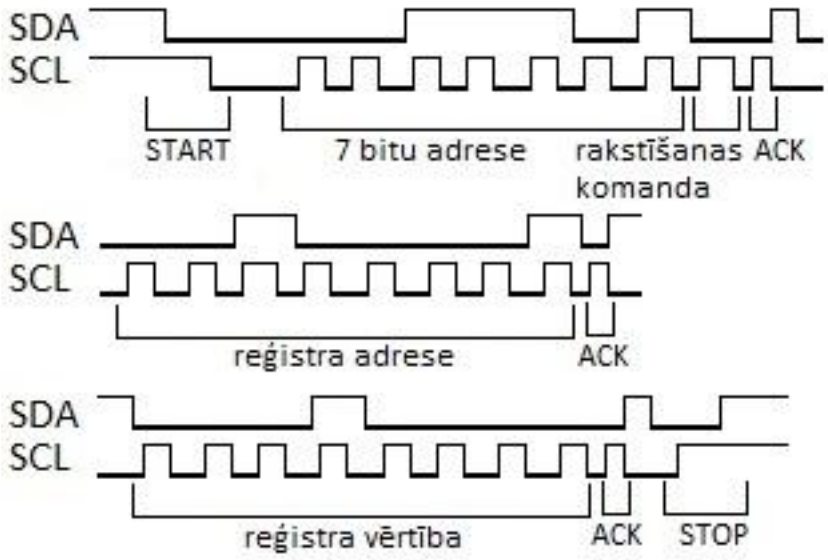
Lai tiktu galā ar šo kļūdu, no sākuma tika pārbaudīts, vai sensors strādā. Tika uzlikts gatavs kods uz ARDUINO un tam klāt pievienots sensors. No šī tika secināts, ka vaina ir sensorā nevis kodā.

Nākamā darbība – tika pārbaudīti visi kontakti sensoram, vai sensors ir kārtīgi pielodēts. Atklājās, ka daži pini nav pielodējušies. Sensors tika kārtīgi pielodēts un rezultātā uz ARDUINO koda tas strādāja.

Pievienojot pie oscilogrāfa tika saņemti šādi dati:



Zemāk tā pati bilde ar skaidrojumiem:



Tika secināts, ka tagad sensors strādā – pēc katra saņemtā bita, nosūta ACK bitu 0.

Bildē ir redzams, kā tiek nosūtīta reset komanda.

Reģistra adrese 00100001 2. Kontroles reģistrs 0x21

Tajā tiek ievadīta vērtība: 00010000 – BOOT 0x10

Tika secināts, ka sensors strādā, kā nepieciešams, un atklāts, ka ar manis rakstīto kodu uz testa platformas sensors sūta nepareizus datus.

Lai izlabotu šo kļūdu, tika pārbaudīts MansOS dotais I2C baitu nolasīšanas un nosūtīšanas kods.

Oriģinālais kods (viss kods netiek ievietots. Tikai tā vietas (atzīmētas ar sarkanu), kur tiiek konstatētas kļūdas):

```
=============
#define I2C_SCL_OUT()  pinAsOutput(SCL_PORT, SCL_PIN)
#define I2C_SCL_HI()   pinSet(SCL_PORT, SCL_PIN)
#define I2C_SCL_LO()   pinClr(SCL_PORT, SCL_PIN)

#define I2C_SDA_IN()   pinAsInput(SDA_PORT, SDA_PIN)
#define I2C_SDA_OUT()  pinAsOutput(SDA_PORT, SDA_PIN)
#define I2C_SDA_HI()   I2C_SDA_IN()
#define I2C_SDA_LO()   I2C_SDA_OUT()
#define I2C_SDA_LO_INIT()   pinClr(SDA_PORT, SDA_PIN)

#define I2C_SDA_GET()  pinRead(SDA_PORT, SDA_PIN)
#define I2C_SDA_SET(b) pinWrite(SDA_PORT, SDA_PIN, b)
```

```
---
//Initializes the ports for I2C
---
#define i2cInit()      \
  I2C_SDA_OUT();       \
  I2C_SCL_OUT();       \
  I2C_SDA_LO_INIT();   \
  I2C_SDA_LO();        \
  I2C_SCL_LO();        \
  I2C_SDA_HI();        \
  I2C_SCL_HI();
```

Kļūdas i2c_soft.h failā tika izlabotas šādi:

```
#define I2C_SCL_OUT()  pinAsOutput(SCL_PORT, SCL_PIN)
#define I2C_SCL_HI()   pinSet(SCL_PORT, SCL_PIN)
#define I2C_SCL_LO()   pinClr(SCL_PORT, SCL_PIN)

#define I2C_SDA_IN()   pinAsInput(SDA_PORT, SDA_PIN)
#define I2C_SDA_OUT()  pinAsOutput(SDA_PORT, SDA_PIN)
#define I2C_SDA_HI()   pinSet(SDA_PORT, SDA_PIN)
#define I2C_SDA_LO()   pinClr(SDA_PORT, SDA_PIN)
#define I2C_SDA_LO_INIT()   pinClr(SDA_PORT, SDA_PIN)

#define I2C_SDA_GET()  pinRead(SDA_PORT, SDA_PIN)
#define I2C_SDA_SET(b) pinWrite(SDA_PORT, SDA_PIN, b)


#define i2cInit()      \
  I2C_SDA_OUT();       \
  I2C_SCL_OUT();       \
  I2C_SDA_LO();        \
  I2C_SCL_LO();        \
  I2C_SDA_HI();        \
  I2C_SCL_HI();
```

Turpinājumā tika skatīts i2c_soft.c fails, tika konstatētas šādas kļūdas:

```
i2cError_t i2cWriteByte(uint8_t txByte)
{
  uint8_t mask;
  i2cError_t err=0;

  for (mask=0x80; mask>0; mask>>=1)   //shift bit for masking (8 times)
  {
    if ((mask & txByte) == 0) {
      I2C_SDA_LO();                  //write a bit to SDA-Line
    }
    else {
      I2C_SDA_HI();
    }
    wait_1us;
    I2C_SCL_HI();
    wait_5us;
    I2C_SCL_LO();
    wait_1us;
```

```
  }
  I2C_SDA_IN();                          //release SDA-line
  I2C_SCL_HI();                          //clk #9 for ack
  wait_1us;
  if (I2C_SDA_GET() != 0) {
    err = I2C_ACK_ERROR;              //check ack from i2c slave
  }
NETIEK IZPILDĪTA KOMANDA I2C_SDA_OUT(); turpmāk nevarēs rakstīt sensorā
  I2C_SCL_LO();
  wait_5us;
  //wait_20us;                       //dely to see the package on scope
  return err;                        //return error code
}



//---------------------------------------------------------------------------
---
// Reads a byte from I2C
// Returns the byte received
// note: timing (delay) may have to be changed for different microcontroller
//---------------------------------------------------------------------------
---
uint8_t i2cReadByte(i2cAck_t ack)
{
  uint8_t mask, rxByte=0;

  I2C_SDA_IN();                          //release SDA-line
  for (mask=0x80; mask>0; mask>>=1)    //shift bit for masking
  {
    I2C_SCL_HI();                        //start clock on SCL-line
    wait_4us;
    if( I2C_SDA_GET() != 0 ){
      rxByte = (rxByte | mask); //read bit
    }
    I2C_SCL_LO();
    wait_1us;                            //data hold time
  }
  if( ack ) I2C_SDA_LO();              //send acknowledge if necessary
IR PADOTA KOMANDA, LAI NOSŪTĪTU ACK, BET NAV PADOTA KOMANDA LAI NOSŪTĪTU NACK
  wait_1us;
  I2C_SCL_HI();                          //clk #9 for ack
  wait_5us;
  I2C_SCL_LO();
  I2C_SDA_IN();                          //release SDA-line
  wait_5us;
  //wait_us(20);                       //dely to see the package on scope
  return rxByte;                         //return error code
}
```

Kļūdas tika izlabotas šādi:

```
  }
  I2C_SCL_LO();
  wait_5us;
  //wait_20us;                        //dely to see the package on scope
I2C_SDA_OUT();
  return err;                         //return error code
}

  if( ack ) {
      I2C_SDA_LO();             //send acknowledge if necessary
  }else {
      I2C_SDA_HI();
  }
  wait_1us;
  I2C_SCL_HI();                         //clk #9 for ack
  wait_5us;
  I2C_SCL_LO();
  I2C_SDA_OUT();                         //sensor release SDA-line
  wait_5us;
  //wait_us(20);                        //dely to see the package on scope
  return rxByte;                        //return error code
}
```

Pēc šīm darbībām, sensors darbojās labi.

Pielikumos 5 – 10 var aplūkot X, Y, Z asu maksimālās un minimālās vērtības.

5 PROGRAMMATŪRAS PIRMKODA FRAGMENTI

```
/*************************************************************
*lis3.h by Rihards Balašs (2011)                             *
*                                                            *
*generic functions for LIS3LV02DQ digital accelerometer      *
*                                                            *
*************************************************************/

#ifndef MANSOS_LIS3_H
#define MANSOS_LIS3_H

//-------register list
enum LIS3Registers
{
	LIS3_OFFSET_X = 0x16,		//loaded at startup
	LIS3_OFFSET_Y =	0x17,		//loaded at startup
	LIS3_OFFSET_Z =	0x18,		//loaded at startup
	LIS3_GAIN_X = 0x19,		//loaded at startup
	LIS3_GAIN_Y = 0x1A,		//loaded at startup
	LIS3_GAIN_Z = 0x1B,		//loaded at startup

	LIS3_CTRL_REG1 = 0x20,		//control register default value 00000111 0x07
	LIS3_CTRL_REG2 = 0x21,		//control register default value 00000000 0x00
	LIS3_CTRL_REG3 = 0x22,		//control register default value 00001000 0x08
	LIS3_HP_FILTER_RESET = 0x23,	//dummy register
	LIS3_STATUS_REG	= 0x27,		//status register default value 00000000 0x00

	LIS3_OUTX_L = 0x28,	//output
	LIS3_OUTX_H = 0x29,	//output
	LIS3_OUTY_L = 0x2A,	//output
	LIS3_OUTY_H = 0x2B,	//output
	LIS3_OUTZ_L = 0x2C,	//output
	LIS3_OUTZ_H = 0x2D,	//output

	LIS3_FF_WU_CFG = 0x30,		//default value 00000000 0x00
	LIS3_FF_WU_SRC = 0x31,		//default value 00000000 0x00
	LIS3_FF_WU_ACK = 0x32,		//dummy register
	LIS3_FF_WU_THS_L = 0x34,	//default value 00000000 0x00
	LIS3_FF_WU_THS_H = 0x35,	//default value 00000000 0x00
	LIS3_FF_WU_DURATION = 0x36,	//default value 00000000 0x00

	LIS3_DD_CFG = 0x38,	//default value 00000000 0x00
	LIS3_DD_SRC = 0x39,	//default value 00000000 0x00
	LIS3_DD_ACK = 0x3A,	//dummy register

	LIS3_DD_THSI_L = 0x3C,		//default value 00000000 0x00
	LIS3_DD_THSI_H = 0x3D,		//default value 00000000 0x00
	LIS3_DD_THSE_L = 0x3E,		//default value 00000000 0x00
	LIS3_DD_THSE_H = 0x3F,		//default value 00000000 0x00

};

enum LIS3Commands
{
	LIS3_ADDRESS_W =0x3A,		//slave address + write command 00111010 0x3A
	LIS3_ADDRESS_R =0x3B,		//slave address + read command 00111011 0x3B

	START =	0x87,			//E7 = 11100111 PD1/PD0/DF1/DF0/ST/Zen/Yen/Xen
					//DF set to 40Hz

	BOOT = 0x10,			//10		=		00010000
```

```
FS/BDU/BLE/BOOT/IEN/DRDY/SIM/DAS
                                        //BOOT = 1, to load default values};


//-------list of functions
uint8_t lis3AddressWrite();
uint8_t lis3AddressRead();
uint8_t lis3WriteRegister(uint8_t regName, uint8_t regValue);
void lis3Reset();
uint8_t lis3Config();
int16_t readRegister16(uint8_t regNameH, uint8_t regNameL);

//-------slave address + write command
uint8_t lis3AddressWrite() {
	uint8_t error = 0;
	i2cStart();
	if((error = i2cWriteByte(LIS3_ADDRESS_W))){
		PRINTF("address write error\n");
	}
	return error;
}
//------- slave address + read command
uint8_t lis3AddressRead() {
	uint8_t error = 0;
	i2cStart();
	if((error = i2cWriteByte(LIS3_ADDRESS_R))){
		PRINTF("address read error\n");
	}
	return error;
}
//-------writes given value in register
uint8_t lis3WriteRegister(uint8_t regName, uint8_t regValue) {
	uint8_t error = 0;
	if((error = lis3AddressWrite())){
		return error;
	}else {
		if((error = i2cWriteByte(regName))) {
			return error;
		}else {
			if((error = i2cWriteByte(regValue))) {
				return error;
			}
		}
	}
	i2cStop();
	return error;
}
//-------reset command
void lis3Reset() {
	if(lis3WriteRegister(LIS3_CTRL_REG2, BOOT)) {
		PRINTF("BOOT FAILED\n");
	}
	mdelay(1);
}
//-------sensor configuration
uint8_t lis3Config() {
	uint8_t error = 0;
	lis3Reset();
	if((error = lis3WriteRegister(LIS3_CTRL_REG1, START))) {
		PRINTF("START failed");
	}
	return error; //return = 0, if ok
}
//-------reads measurment values
int16_t readRegister16(uint8_t regNameH, uint8_t regNameL) {
```

```
	int8_t high=0;
	uint8_t low=0, error = 0;
	int16_t result=0;

	//-------reads first byte
	if((error = lis3AddressWrite())) {
		return error;
	}else {
		//-------address to read
		if((error = i2cWriteByte(regNameH))) {
			return error;
		}else {
			//-------send repeated start + read command
			if((error = lis3AddressRead())) {
				return error;
			}else {
				high = i2cReadByte(I2C_NO_ACK); //read the register
				i2cStop();
			}
		}
	}
	mdelay(100);
	if(error) {
		PRINTF("first byte read error\n");
		return error;
	}
	//-------reads second byte
	if((error = lis3AddressWrite())) {
		return error;
	}else {
		//-------address to read
		if((error = i2cWriteByte(regNameL))) {
			return error;
		}else {
			//-------sends repeated start + read command
			if((error = lis3AddressRead())) {
				return error;
			}else {
				low = i2cReadByte(I2C_NO_ACK); //read the register
				i2cStop();
			}
		}
	}
	if(error) {
		PRINTF("second byte read error\n");
	}
	result = (((int16_t)(high<<8)) + (uint16_t)low);
	return result;
}

#endif
```

6 PROJEKTA ORGANIZĀCIJA

Digitālā akselerometra dzinis tika izstrādāts, par pamatu ņemot ūdenskrituma modeli.

Sākumā tika veikta teorētiskā materiāla izpēte un galveno prasību izvirzīšana. Turpinājumā - projektēšana, kuras rezultātā tika izveidota vēlamā testa platforma. Nākamie soļi tika veikti, lai testa platformu atkļūdotu.

Projektēšanas rezultātā tika izveidota vēlamā datu pārraides shēma un moduļu saraksts; tie, izdarot atbilstošus precizējumus programmatūras prasību specifikācijā, tika secīgi, detalizēti projektēti un implementēti. Visbeidzot tika veikta sensora sagatavošana, programmatūras testēšana un produkta dokumentācijas galīgās versijas izstrāde. Atklātās kļūdas tika uzreiz izlabotas.

Nepieciešamo teorētisko materiālu iegūšanā un pamatprasību definēšanā autoram palīdzēja darba vadītājs. Turpmākā projekta izstrāde tika veikta patstāvīgi. Darba autors veica projektētāja, programmētāja, dokumentācijas izstrādātāja funkcijas. Testa platformas un programmatūras koda testēšanas laikā autoram palīdzēja darba vadītājs.

Projekta izstrāde notika, izmantojot MansOS C kompilatoru, GNU „Kate" teksta redaktoru.

7 KONFIGURĀCIJU PĀRVALDĪBA

Programmatūras produkta un dokumentācijas konfigurāciju pārvaldībai netiek izmantota nekāda papildus programmatūra.

Projekta kods kopā ar konfigurācijas repozitoriju tika glabāts uz izstrādes datora cietā diska partīcijas, kā arī katras izstrādes dienas beigās tika veidota rezerves kopija uz Windows operētājsistēmas partīcijas un USB flešatmiņas.

8 KVALITĀTES NODROŠINĀŠANA

Lai nodrošinātu kvalitatīva produkta izstrādi, tika veiktas šādas darbības:

- Produkta dokumentācija tika izstrādāta vadoties pēc atbilstošiem Latvijas Valsts standartiem;
- Lai nodrošinātu koda noformējuma konsistenci, tika izpētīti MansOS faili. Tika rakstīts kods, kas ir atbilstošs MansOS pieņemtajam stilam;
- Produkta kods tika pilnībā komentēts angļu valodā. Angļu valodā tika komentēti metožu nolūki, sarežģītākajām metodēm tika veidoti komentāri arī par to iekšējo darbību;
- Svarīgākajām metodēm tika pievienoti cikla izvades komentāri. Redzot kļūdas iemeslu, tika atvieglota testēšana.

9 DARBIETILPĪBAS NOVĒRTĒJUMS

Te tiek uzskaitīti darbi, kas tika veikti, lai realizētu projektu:

- Iepazīšanās ar shēmu veidošanas, projektēšanas procesu;
- Nepieciešamo komponenšu saraksta izveide;
- Komponenšu simbolu zīmēšana;
- Shēmas zīmēšana;
- Testa platformas dizaina izveidošana;
- Testa platformas pasūtīšana;
- Pasūtītās testa platformas komponenšu lodēšana;
- Testa platformas atkļūdošana;
- Iepazīšanās ar I2C protokolu;
- Sensora piemeklēšana;
- Sensoram nepieciešamās shēmas izveide;
- Sensora lodēšana uz shēmas;
- Sensora dziņa izveide;
- Sensora un tā dziņa testēšana, atkļūdošana;
- Dokumentācijas veidošana;

Šīs darbības tika veiktas 3 mēnešu garumā. Ilgāko laika posmu tika strādāts, lai izveidotu testa platformu, iepazītu I2C protokolu, kā arī lai produktu testētu.

10 REZULTĀTI

Darba izstrādes gaitā ir izveidota ērta un funkcionējoša testa platforma Tmote Mini bezvadu sensora modulim, kas atbilst izstrādātajai prasību specifikācijai. Tika veikta visas izstrādes dokumentēšana un izstrādājamā produkta testēšana. Visa izstrāde notika atbilstoši valsts un starptautiskajiem standartiem.

Darba izstrādes gaitā autors iepazinās ar projekta izstrādes dzīves cikla posmiem un apguva jaunas iemaņas šādās jomās:

- C programmēšanā;
- Mikrokontrolieru programmēšanā;
- Mikroshēmu veidošanā (projektēšanā, kodināšanā, atkļūdošanā).

Izveidotā testa platforma piedāvā nepieciešamās iespējas - nodrošināt darbu programmatūras speciālistam bez īpašas nepieciešamības veidot atsevišķu platformu vajadzīgajiem sensoram.

11 SECINĀJUMI

Pēc darba paveikšanas autors secināja, ka izveidotajā testa platformā var ieviest vairākus uzlabojumus. Tie tiks veidoti nākamajā testa platformas versijā.

Viena no problēmām darba gaitā bija sensora sagatavošana darbam. Lai to panāktu vajadzēja projektēt jaunu plati, kuru bija nepieciešams izkodināt. Tad, kad plate bija gatava, autors saskārās ar problēmu, ka izvēlētais sensors ir ļoti maza izmēra, tādēļ pirmo reizi tas netika pielodēts kārtīgi. Atkļūdošanas procesā šīs kļūdas tika izlabotas.

Nākotnes plānos ietilpst otrā testa platformas versija, kurai nebūs vajadzības manuāli pārslēgties starp programmēšanas un seriālā porta klausīšanās režīmiem. Ir plānots uz testa platformas uzlikt papildus čipu, kas ar īpaši veidotas programmas palīdzību automātiski pārslēgs vajadzīgo režīmu.

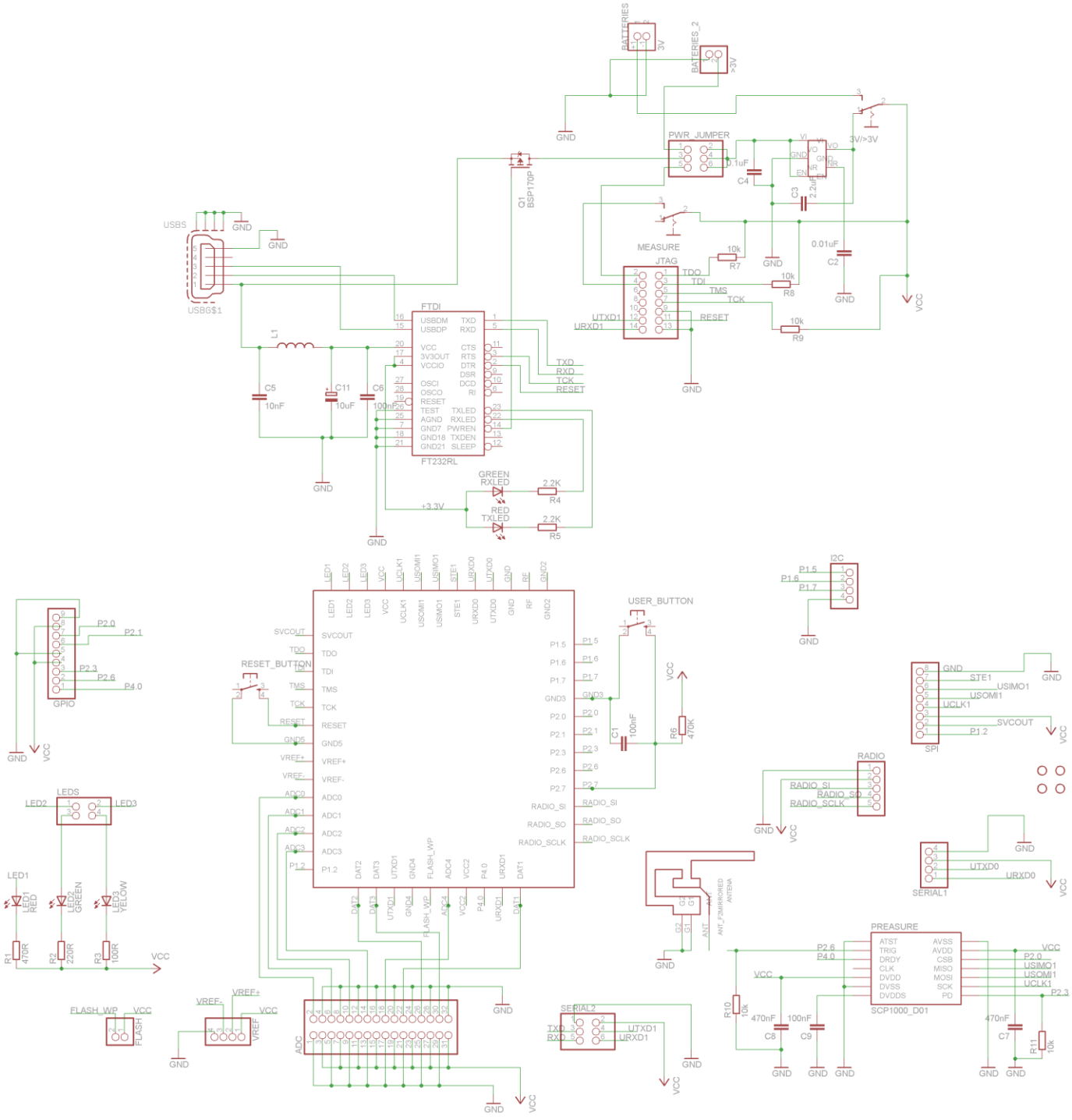
12 IZMANTOTĀ LITERATŪRA UN AVOTI

1) Sparkfun EagleCad pamācība 1. daļa [skatīts 2011.02.05]
   http://www.sparkfun.com/tutorials/108
2) Sparkfun EagleCad pamācība 2. daļa [skatīts 2011.02.06]
   http://www.sparkfun.com/tutorials/109
3) Sparkfun EagleCad pamācība 3. daļa [skatīts 2011.02.06]
   http://www.sparkfun.com/tutorials/110
4) C programmēšanas valodas operātori [skatīts 2011.03.12]
   http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B
5) FTDI specifikācija [skatīts 2011.02.16]
   http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
6) Tmote Mini specifikācija [skatīts 2011.02.20]
   http://sentilla.com/files/pdf/eol/Tmote_Mini_Datasheet.pdf
7) Digitālā akselerometra LIS3LV02DQ specifikācija [skatīts 2011.03.15]
   http://www.google.lv/url?sa=t&source=web&cd=1&ved=0CCUQFjAA&url=http%3A%2F%2Fwww.symbrion.eu%2Ftiki-download_file.php%3FfileId%3D225&rct=j&q=lis3lv02dq&ei=du3cTbi3IcmXOrrHufcO&usg=AFQjCNGpiRvY5N9HEgGruZIKk15mpxZhpA&sig2=mv05lKbBFFJtFKTIobcCXA&cad=rja
8) I2C protokola pamācība [skatīts 2011.03.20]
   http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

13 PIELIKUMI

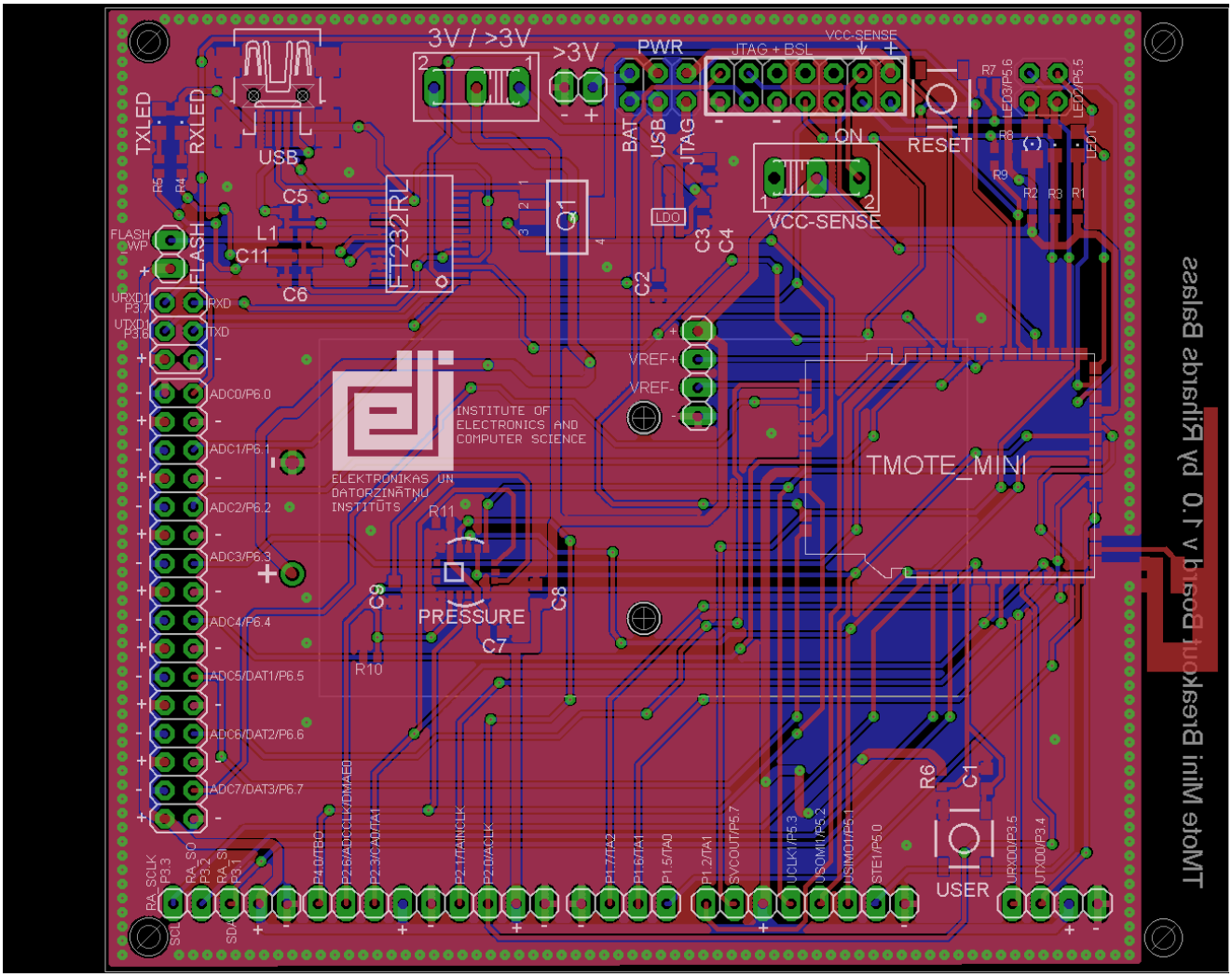
13.1 Pielikums 1

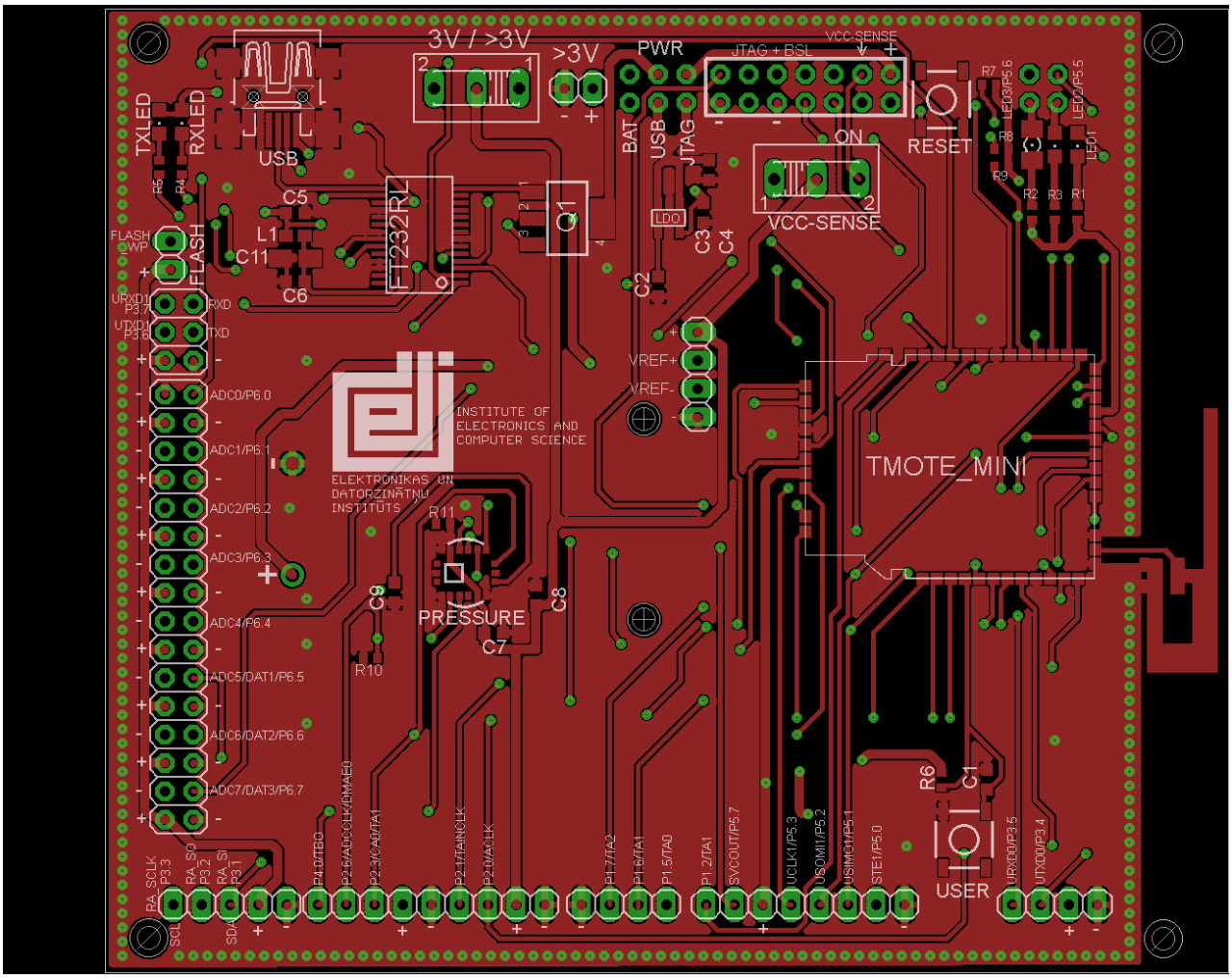
Testa platformas shēma



13.2 Pielikums 2

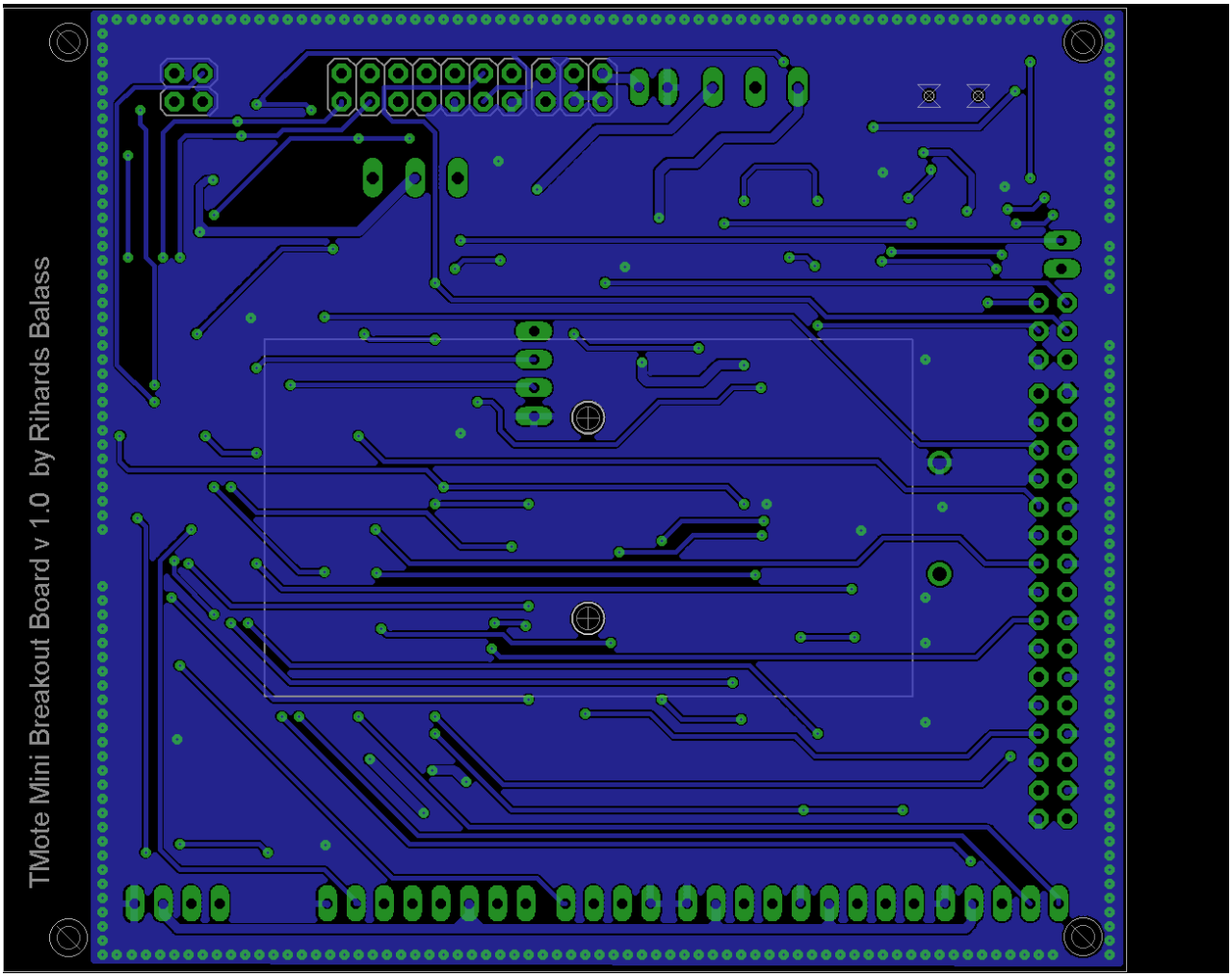
Testa platformas dizaina abas puses




3V / >3V
>3V
PWR
JTAG + BSL
VCC-SENSE
RESET
ON
VCC-SENSE
TXLED
RXLED
USB
FT232RL
FLASH
L1
C11
C5
C6
C1
C2
C3
C4
LDO
BAT
USB
JTAG
VREF+
VREF-
INSTITUTE OF
ELECTRONICS AND
COMPUTER SCIENCE
ELEKTRONIKAS UN
DATORZINĀTNU
INSTITŪTS
TMOTE_MINI
PRESSURE
R11
R10
C7
C8
C9
R6
USER
ADC0/P6.0
ADC1/P6.1
ADC2/P6.2
ADC3/P6.3
ADC4/P6.4
ADC5/DAT1/P6.5
ADC6/DAT2/P6.6
ADC7/DAT3/P6.7
RXD
TXD
FLASH WP
LED3/P5.6
LED2/P5.5
LED1

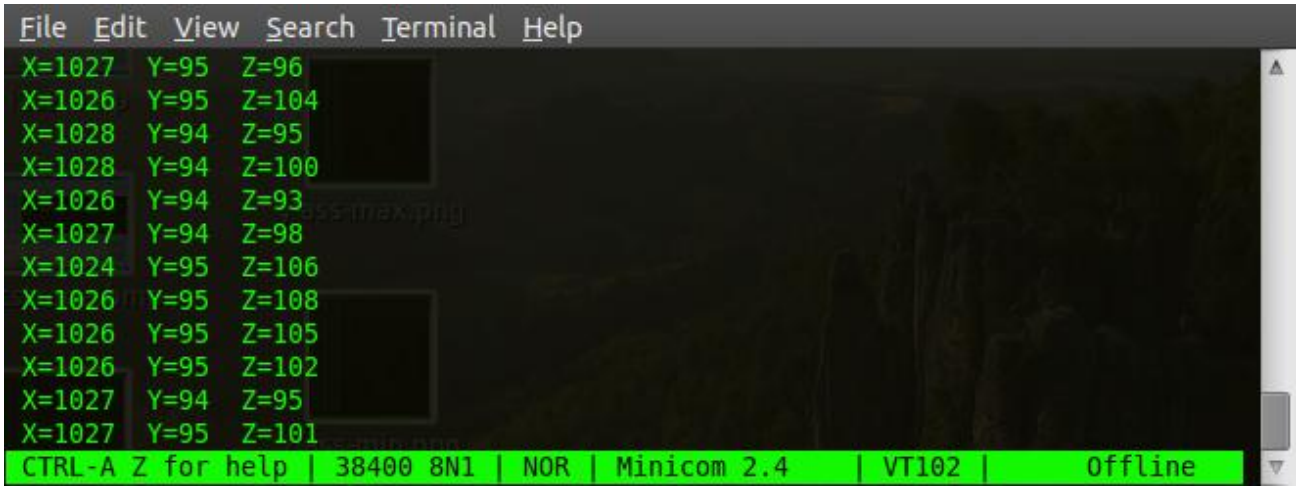
13.4 Pielikums 4

Testa platformas dizaina apakšpuse



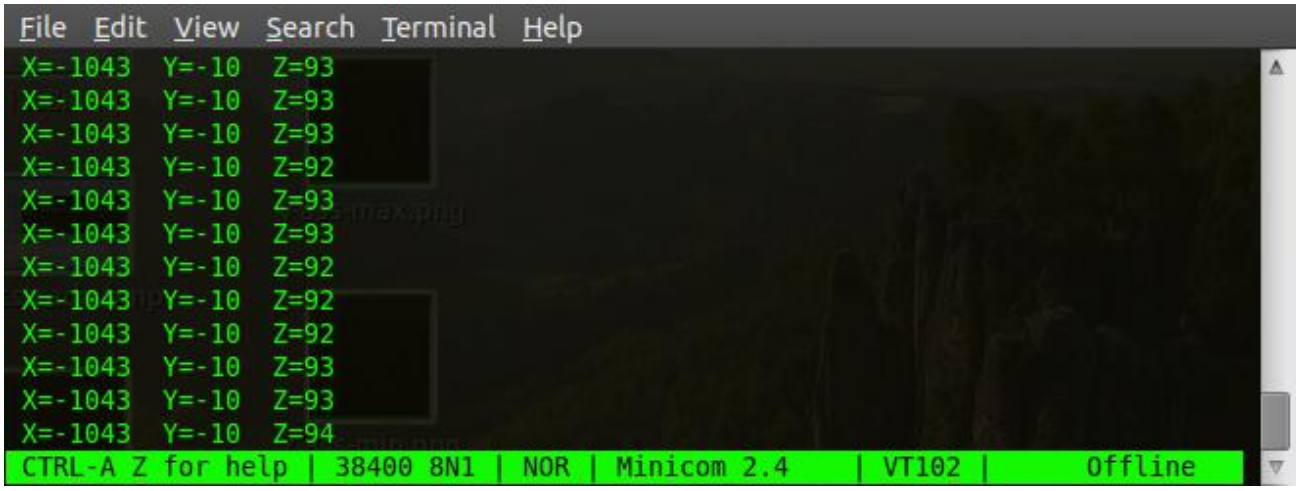
13.5 Pielikums 5

X ass maksimālā vērtība



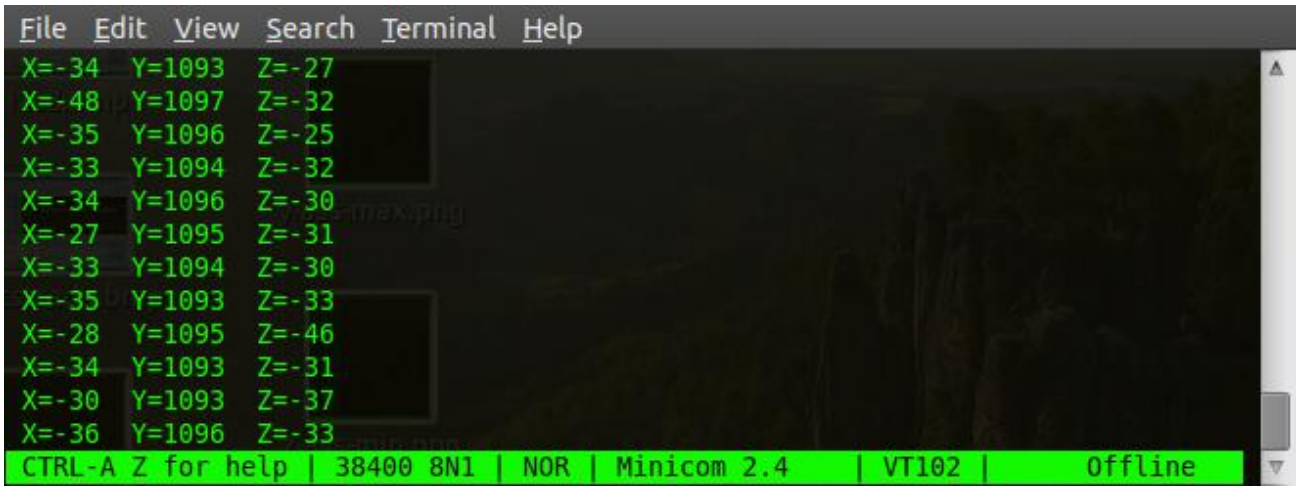
13.6 Pielikums 6

X ass minimālā vērtība



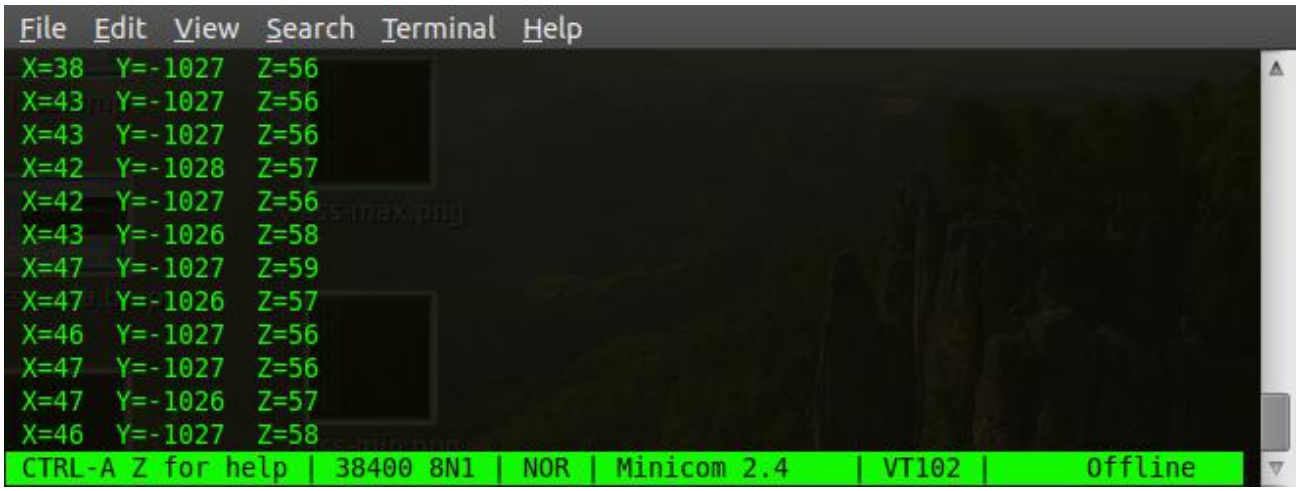
13.7 Pielikums 7

Y ass maksimālā vērtība



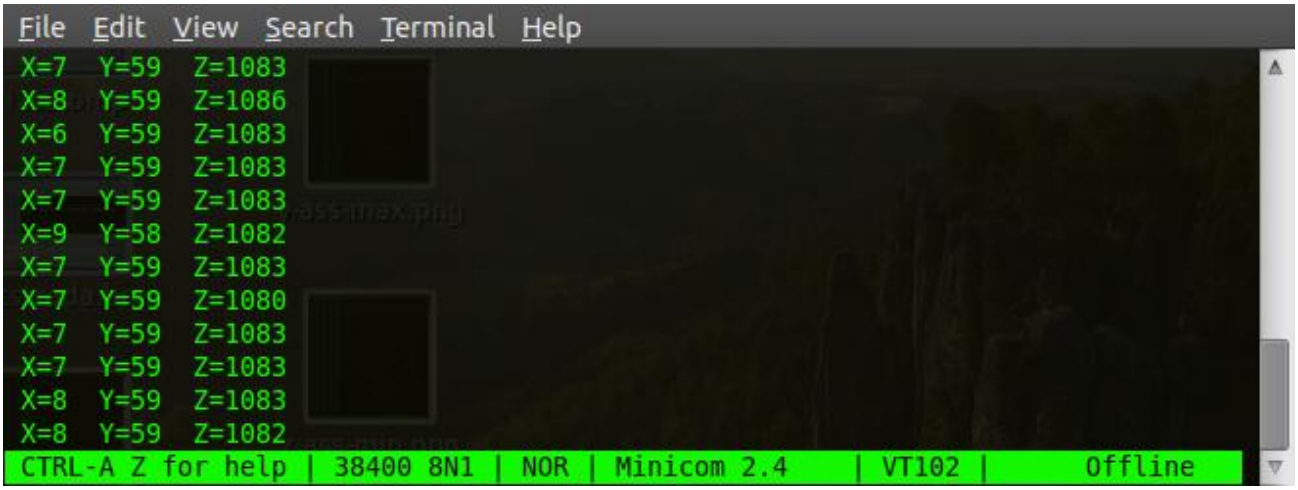
13.8 Pielikums 8

Y ass minimālā vērtība



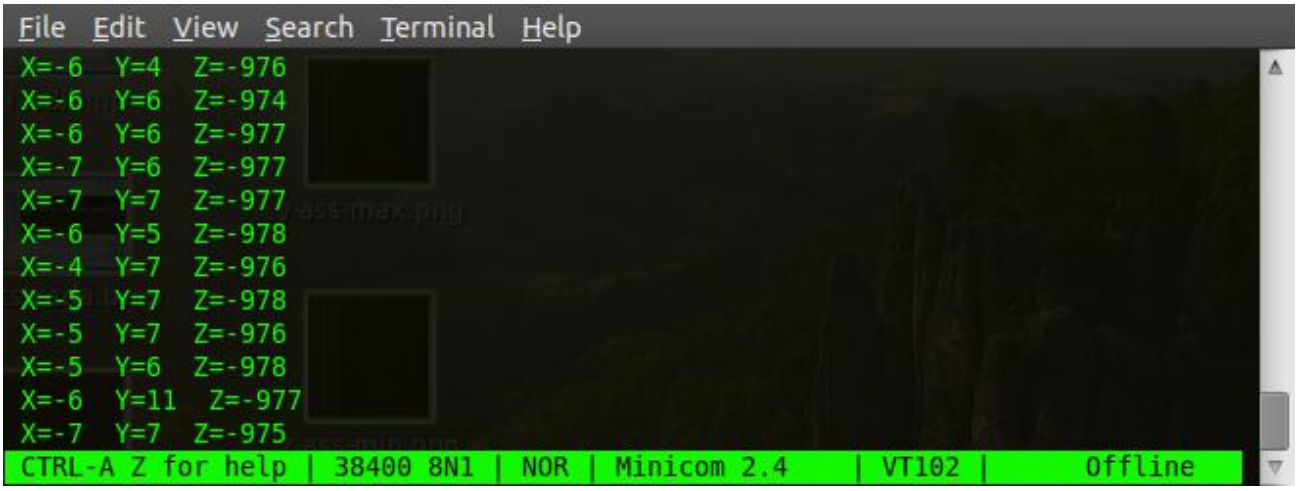
13.9 Pielikums 9

Z ass maksimālā vērtība



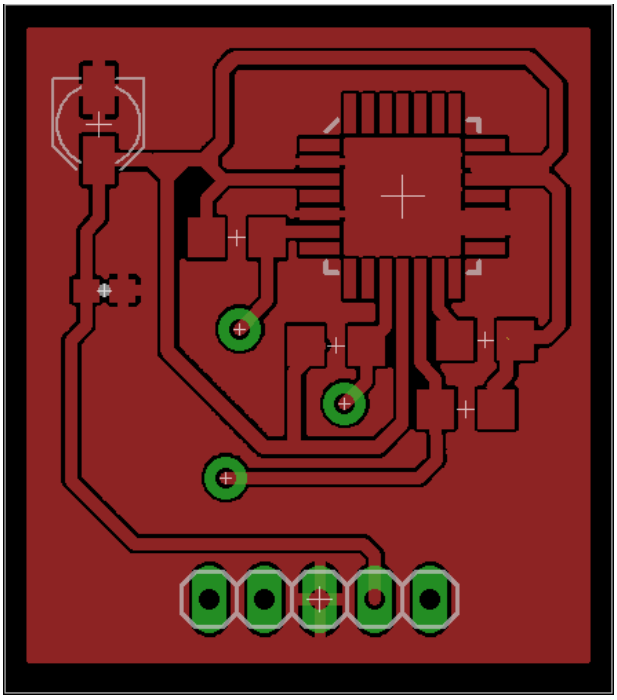
13.10 Pielikums 10

Z ass minimālā vērtība



13.11 Pielikums 11

Akselerometra plate

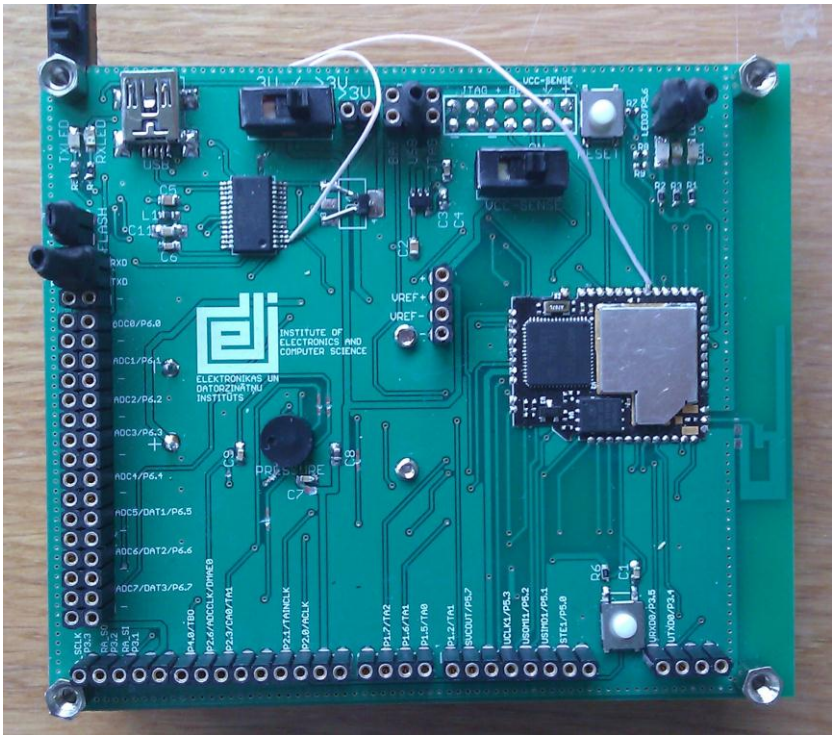


13.12 Pielikums 12

Akselerometra plate reāli

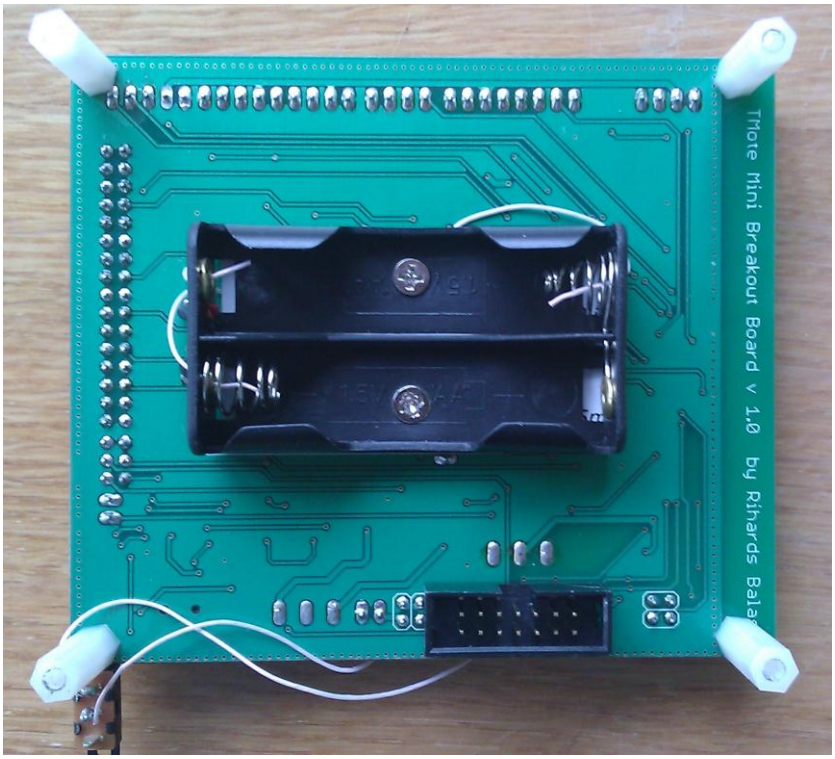
13.13 Pielikums 13

Testa platforma reāli - augšpuse



13.14 Pielikums 14

Testa platforma reāli - apakšpuse



Kvalifikācijas darbs „***Testa platforma Tmote Mini bezvadu sensora modulim***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ***Rihards Balašs*** _____ **___.05.2011**.

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: ***Dr.dat Leo Seļāvo*** _____ **___.05.2011.**

Recenzents: ***Dr.dat Juris Rāts***

Darbs iesniegts 26.05.2011.
Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Dainis Dosbergs* _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
___.06.2011. prot. Nr. _____, vērtējums _____ (_____)

Komisijas sekretārs(-e): _____

Recenzenta atsauksme
par LU Datorikas fakultātes pirmā līmeņa profesionālās augstākās izglītības studiju programmas „Programmēšana un datortīklu administrēšana” specializācijā SE (Programmētājs)

studenta Riharda Balaša
kvalifikācijas darbu „Testa platforma Tmote Mini bezvadu sensora modulim”

	vērtējums[1]
Ir iekļauta pretendenta izstrādātajam programmatūras produktam atbilstoša prasību specifikācija un projektējums (vai to fragmenti), kas drīkst nebūt paša pretendenta izstrādāti	
Piezīmes:	
Ir iekļauts pretendenta patstāvīgi izstrādāts zema līmeņa (datu struktūru un algoritmu) projektējums	
Piezīmes:	
Ir iekļauts pretendenta patstāvīgi izstrādāts un labi komentēts programmas kods	
Piezīmes:	
Ir izstrādāta testēšanas dokumentācija, kas apliecina pretendenta patstāvīgi veiktu vienībtestēšanu	
Piezīmes:	
Ir iekļauts paskaidrojošs teksts, kurā atspoguļota konkrētā programmatūras projekta: 1) organizācija	
2) kvalitātes nodrošināšana	
3) konfigurāciju pārvaldība	
4) darbietilpības novērtējums	
Piezīmes:	
Vai datorprogramma atbilst vismaz 3 personmēnešu darbietilpībai?	
Vai programmatūras produkts ir noformēts saskaņā ar valsts vai starptautiskiem standartiem?	
Vai programmprodukta izstrāde notikusi saskaņā ar labo programmēšanas praksi?	
Citas piezīmes par darbu:	

	Vērtējums (0 – 10)
Darba noformējuma kvalitātes novērtējums	
Programmatūras produkta novērtējums	
Recenzenta galīgais vērtējums	

Recenzents Dr.dat Juris Rāts_____
(paraksts, datums)

[1] Ar vērtībām JĀ / NĒ / DAĻĒJI