

LIBELIUM WASPMOTE bezvadu sensoru tīkli.

Laboratorijas darbi

I. IEVADS

Laboratorijas darbi tiks veikti uz Libelium waspmote platēm (skatīt attēls I.), kā arī tiem tiks pieslēgti arī citi papildus moduļi apkārtējās vides pētīšanai un komunikācijai. Paša moduļa specifikācija ir aplūkojama tabulā. Pielikums un Pielikums ir aplūkojamas moduļa svarīgākās sastāvdaļas, kuras tiks pielietotas pildot laboratorijas darbus. Savukārt Pielikums var aplūkot uz moduļa izvadītās pieslēgvietu pinu nozīmes, jo pie tām arī tiks vienotas klāt dažādas papildus plates atbilstoši veicamajam laboratorijas darbam.



attēls I. Waspote plate ar papildmoduļiem (Xbee un GPRS)

Specifikācijas:

tabula

Mikrokontrolieris	Atmega1281	Barošanas spriegums	3.3V - 4.2V
Frekvence	8MHz	Darbības temperatūra	-20°C, +65°C
SRAM	8kB	Ievad/izvad līnijas	7x analogās
FLASH	128kB		8x digitālās
EEPROM	4kB		1x PWM
Strāvas patēriņš	Normāla darbība: 9mA		2x UART
	SLEEP režīms:		1x I2C
			1xUSB

62uA DEEP SLEEP režims: 62uA HIBERNATE režims: 0.7uA	
---	--

Programmēšana

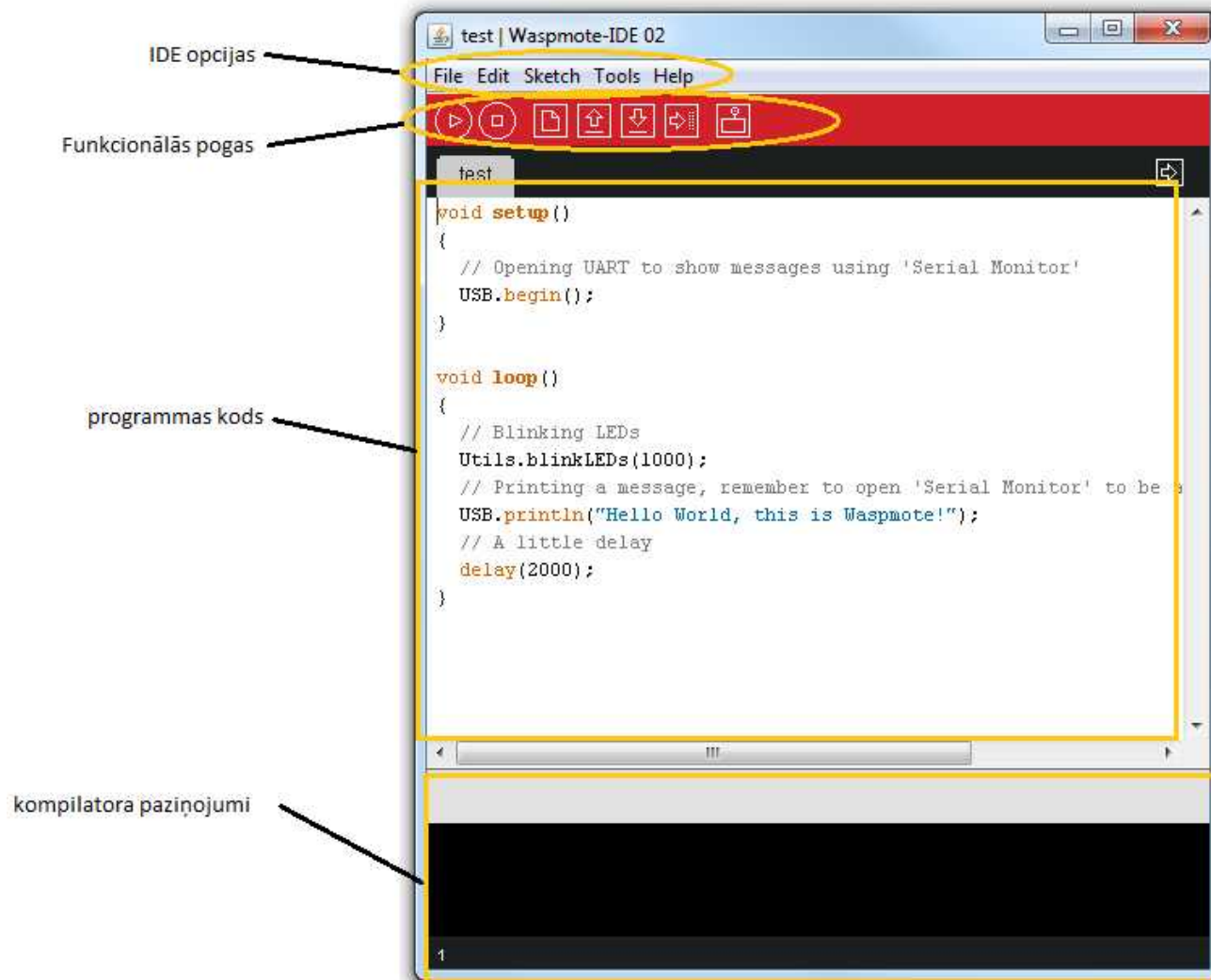
Wasmote moduļu programmēšana notiek C++ valodā, izmantojot Libelium piedāvāto Wasmote IDE programmēšanas vidi, kura jau ir sagatavota gan programmēšanai, gan programmas koda ierakstīšanai uz mikrokontrolieri. Otrs programmēšanas veids ir izmantojot Atmel Studio 6.0, kuram vēl ir nepieciešams papildus veikt projekta uzstādījumus dotajai waspmote platei. Izmantojot WASPMOTE IDE programmēšanas vidi, programma pamatā sastāvēs no divām daļām, *setup* un *loop*. *Setup* daļā tiek inicializēti visi darbam nepieciešamie moduļi un uzstādīti kaut kādi sākotnējie parametri. *Loop* daļā atradīsies pats galvenais programmas kods, kurš tiks izpildīts visu laiku, kad plate ir ieslēgta. Turpretī ja izmanto ATMEL Studio 6.0 programmēšanas vidi, tad parādās arī trešā daļa, *main*. Šajā daļā arī notiek pašas programmas izpildīšanās.

```
int main(void)
{
    init();// tiek inicializēts mikrokontrolieris
    setup();//tiek inicializēta pati plate
    for(;;)//mūžīgais cikls
        loop();//galvenās funkcijas izsaukšana
}
```

Lai inicializētu kādu no plates moduļiem, piemēram komunikāciju caur USB kabeli, nepieciešams *setup* daļā inicializēt USB moduli ar komandu *USB.begin()*; Citiem moduļiem, kurus ir iespējams pievienot, papildus inicializēšanai tos ir nepieciešams ieslēgt, piemēram, *WIFI.begin(); WIFI.ON(socket0);* kur *.begin()*; funkcija inicializē pievienoto WiFi moduli, bet *.ON(socket0);* norāda, ka modulis ir pieslēgts pie pieslēgvietas nr.0 un ieslēdz to. Pēc noklusējuma daži no pievienojamajiem moduļiem var atrasties SLEEP režīmā, tāpēc tos arī vēl ir nepieciešams iespēgt.

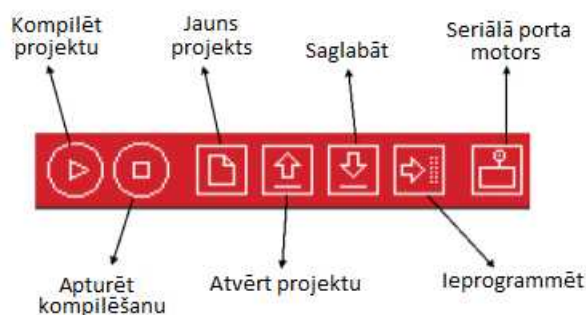
Darba uzsākšana ar Waspmate-IDE programmēšanas vidi

- 1) Atvērt „Waspmate” programmēšanas vidi. Ja ikona nav redzama uz ekrāna, tad programmu var palaist no mapes „C:\wasmate-ide-v.02-windows”. Ja gadījumā tādas mapes nav, tad programmu ir nepieciešams lejupielādēt no <http://www.libelium.com/downloads/wasmate-ide-v.02-windows.zip>



attēls I. programmēšanas vide

Waspmate-IDE opcijas ļauj mainīt dažādus uzstādījumus, piemēram, izvēlēties attiecīgo „COM port” pie kura ir pieslēgts Waspmate plate un pašas plates versiju.



attēls I. Programmas pogas un to funkcijas

Pogas ļauj ātri veikt pašas nepieciešamākās darbības ar porgrammu.

2) Vienkārša programmas koda ielādēšana noteikumi:

a. Iekopēt doto programmas kodu WASPMOTE IDE programmas logā

- Kods:

```
void setup()
{
  USB.begin();
}
void loop()
{
  Utils.blinkLEDs(1000);
  USB.println("Hello World, this is Waspote!");
  delay(2000);
}
```

b. Nokompilēt programmas kodu. Pārliedzināties, vai programmas kodā nav kļūdu. (kompilators parādīs ja būs)



Done compiling.

Binary sketch size: 2498 bytes (of a 122880 byte maximum)

c. Ja programmas kodā nav kļūdu, tad kompilatoru paziņojumu logā jāparādās Error: Reference source not found attēlotajam ziņojumam

d. No Waspote plates jāatvieno Xbee radio modulis, jo tas izmanto to pašu interfeisu, caur kuru tiek programmēta pati plate.

e. Pievienot plati pie datora izmantojot mini USB kabeli.

f. Ieslēgt Waspote plati. ON/OFF slēdzis jānobīda tālāk no mini USB konektora.

g. WASPMOTE IDE programmā jānorāda COM ports pie kuram waspote plate ir pieslēgta

- Atvērt „device manager” (START → RUN → “devmgmt.msc” → OK)

- Apskatīt pieejamos COM portus [Ports(COM & LPT)] un nosakam, pie kura COM porta ir pieslēgtam waspote plate.

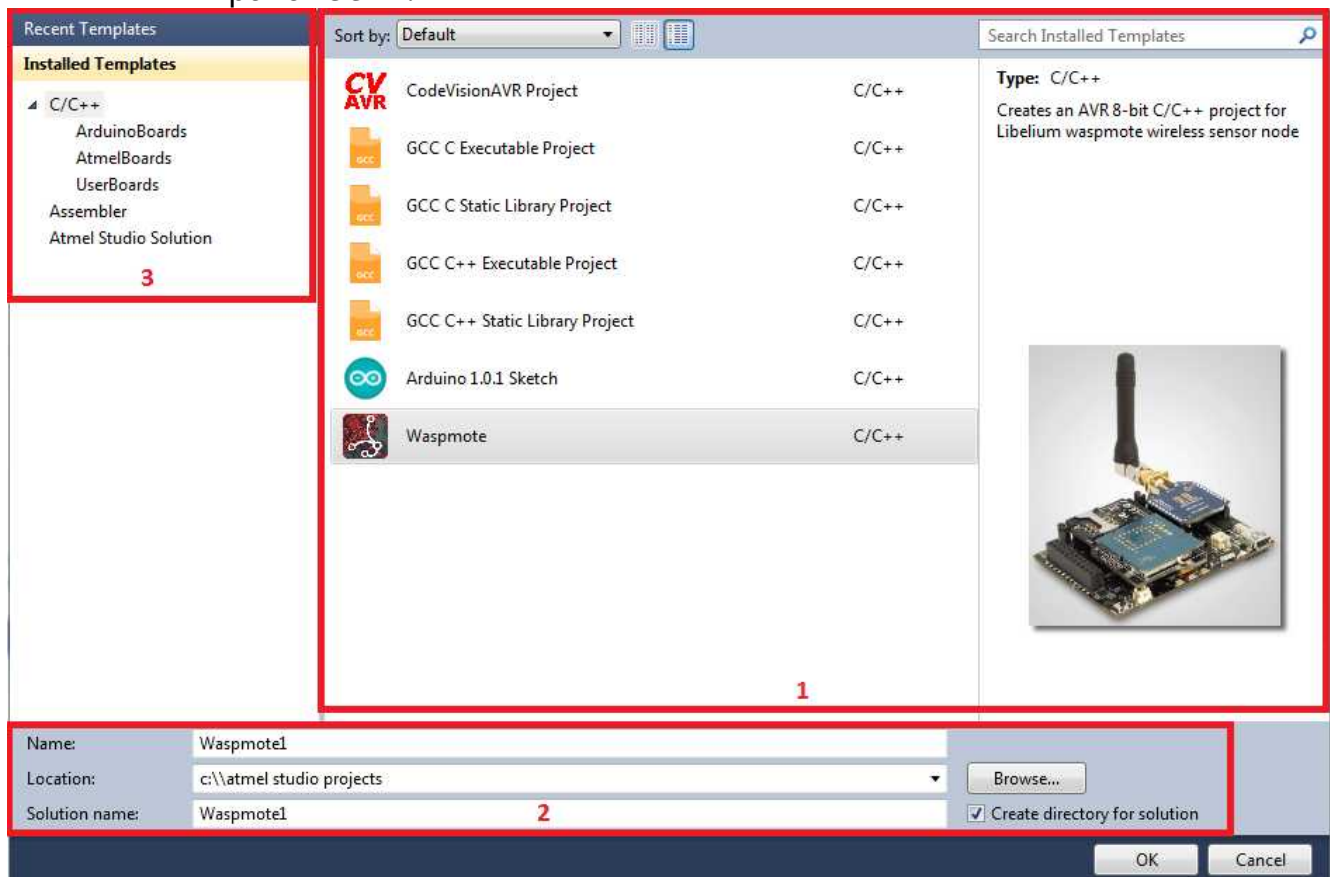
- WASPMOTE-IDE programmā norādam attiecīgo COM portu [Tools → Serial Port → {izvēlamies attiecīgo COM portu}]. Pie reizes vēl arī pārliedzināties vai [Tools → Board] izvēlnē ir izvēlēta „waspote-api-v.xxx”, kur ‘x’ ir aizstāts ar cipariem, kuri norāda platei izmantojamo bibliotēku versiju.

h. Spiežam ieprogrammēt programmas kodu uz waspote plates.

i. Pēc kāda laiciņa kompilatora paziņojumu logā būtu jāparādās „Done uploading” ziņojumam. Uz plates var novērot, kā mirgo divas gaismas diodes, zaļa un sarkana. Atverot seriālā porta monitoru, parādās šādi ziņojumi: „Hello World, this is Waspote!”. Tas nozīmē, ka viss izpildīts pareizi un veiksmīgi.

Darba uzsākšana ar ATMEL Studio 6.0 programmēšanas vidi

- 1) Palaist ATMEL Studio 6.0 programmu.
- 2) Pēc programmas palaišanas parādās „Start Page” logs, kurā var izvēlēties veidot jaunus projektus vai atvērt iepriekšējos projektus no sadaļas „Recent Projects”. Katram laboratorijas darbam būs nepieciešams izveidot jaunu projektu:
 - a. „Start Page” logā spiežam „New Project...”.
 - b. Parādās jauns logs, kurā ir jāizvēlas jaunā projekta parametri:



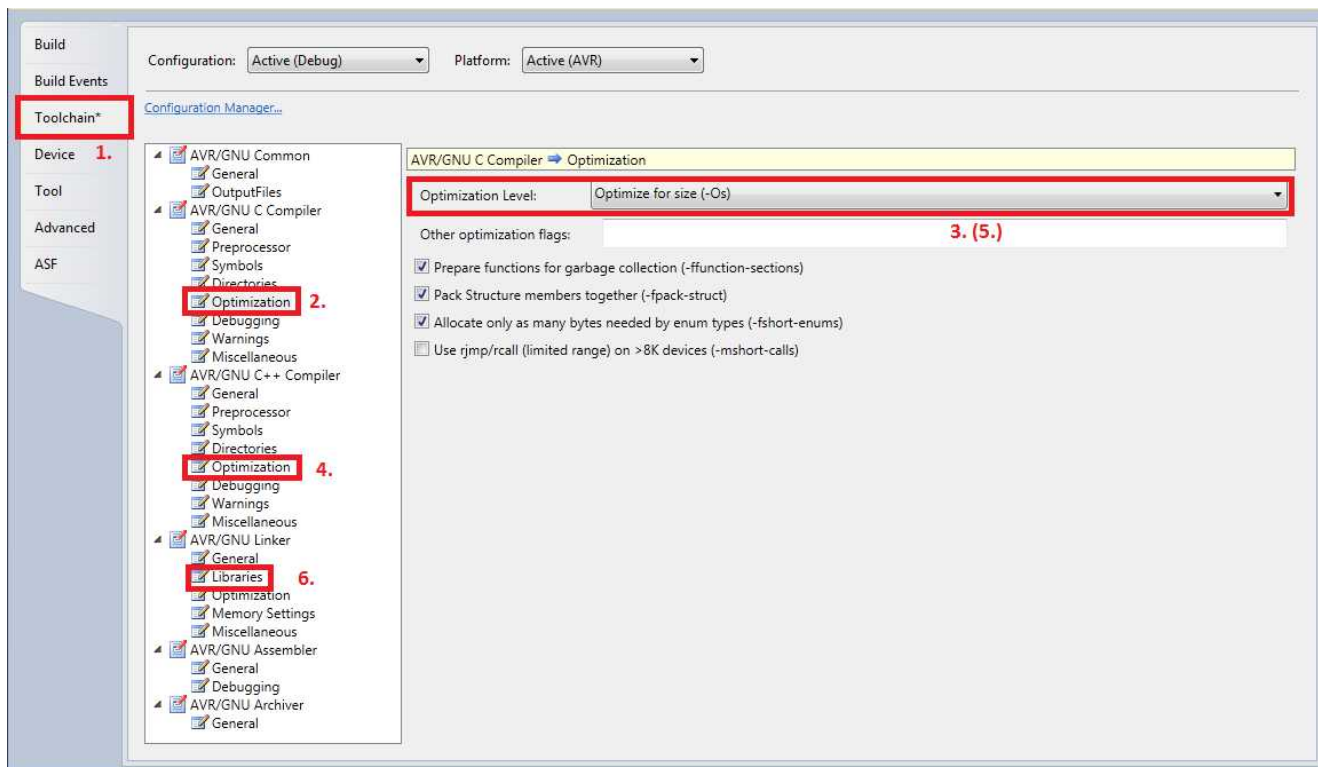
Dotais logs ir sadalīts trijās daļās: Pirmajā daļā tiek izvēlēts projekta veids, kam dotais projekts tiek veidots un uz kādas bāzes. Otrajā daļā var izvēlēties paša projekta nosaukumu un vietu, kur projekts tiks saglabāts. Trešajā daļā var izvēlēties jau ATMEL un citu ražotāju piedātos projektus kādai konkrētai platformai. Izvēlēties jauno projektu bāzētu uz Libelium Waspote.



Waspote

- 3) Parādās jauns logs, kurā ir redzams sākotnējais programmas kods. Diemžēl Atmel Studio 6.0 nav ideāla un tai arī ir savas kļūdas (Cerams ka ražotāji tās novērs). Tāpēc šajā solī jau izveidotajam projektam ir jāveic papildus uzstādījumi:

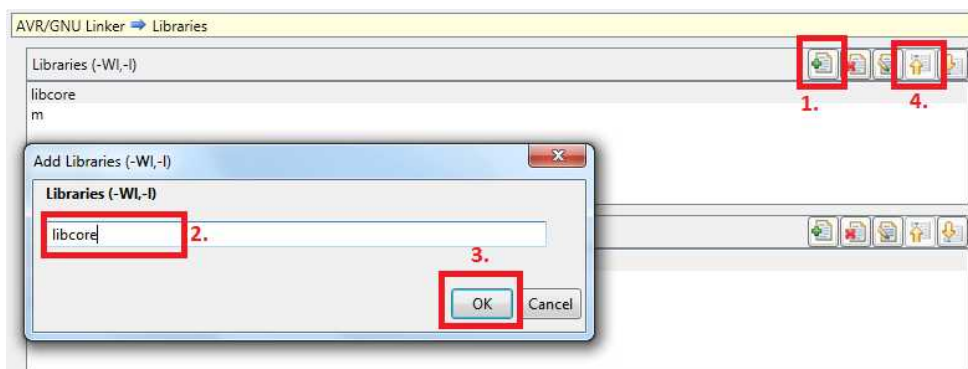
a. Ir jāatver projekta opciju logs: spiežam „Alt + F7”.
Jaunatvērtā loga kreisajā malā ir jāizvēlas „Toolchain”




sadaļa.

b. Kā parādīts attēls I., sarakstā izvēlamies sadaļu „AVR/GNU C Compiler ➤ Optimization” un blakus logā izmainām „Optimization Level” un „Optimize for size (-Os)”

c. Tā pat arī ar „AVR/GNU C++ Compiler ➤ Optimization”
d. „AVR/GNU Linker ➤ Libraries” sadaļā „Libraries (-WI, -I)” pievienojam klāt ierakstu „libcore”, kā parādīts attēls I..



4) Iekopējam to pašu programmas kodu, kurš dots iepriekš 2.a punktā.

5) Pārliedzināties vai programmas kods ir pareizs un kompilējam to. Spiežam taustiņu „F7” vai ātrās izvēlnes joslā  pogu.

6) Ja programmas kods ir pareizs, tad jāparādās „Output” logā „Build succeeded” ziņojumam.

7) Kā ar WASPMOTE IDE programmu, tāpat arī Atmel Studio programmā ir jānorāda, pie kura COM porta ir pieslēgta waspmote plate:

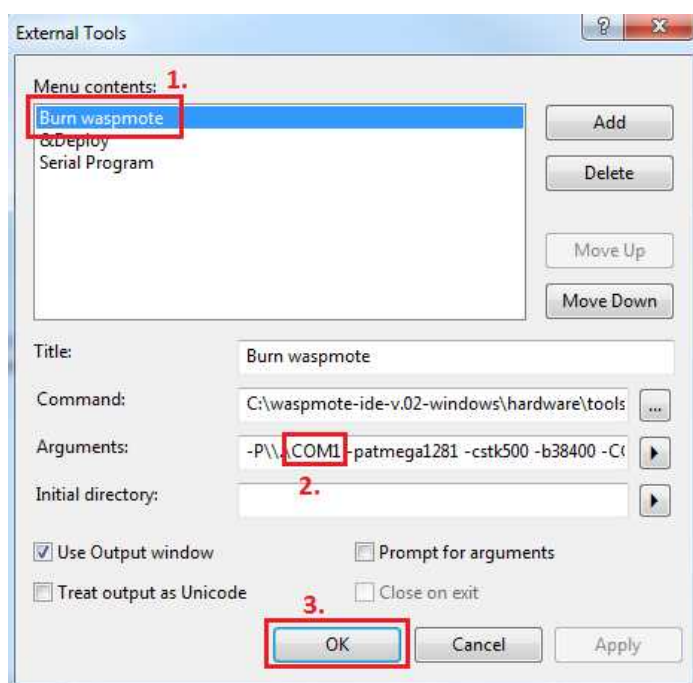
a. Tools ⇒ External Tools...

b. Jaunajā logā izvēlamies Burn waspmote

c. „Argumets:” rindā norādām attiecīgo COM portu (piem., „...-P\\.\COM1 ...”)

Viss pārējais paliek bez izmaiņām

8) Tagad atliek programmas kodu ierakstīt waspmote platē



[Tools ⇒ Burn waspmote].

II. Laboratorijas darbi

2. laboratorijas darbs. Libelium plates pamatmoduļu izmantošana

Darba uzdevumi:

- RTC modulī ieprogrammēt šodienas datumu un laiku;
- Izvadīt
- Uzprogrammēt zaļo LED gaismas diodi mirgot reizi sekundē, un sarkano gaismas diodi mirgot gadījumos, kad notiek komunikācija caur UART-USB interfeisu.
- * Uz datora katru sekundi attēlot datumu, laiku un akselerometra mērījumus.

Apraksts:

Laboratorijas darbā ir jāapgūst Libelium waspmote plates pamatelementu izmantošanu. Šo pamatelementu pielietošana būs nepieciešama nākošo laboratorijas darbu veikšanai. Sākumā nepieciešams iepazīties ar pašu waspmote plati un kā to programmēt, kā katru moduli inicializēt un kā izmantot.

Libelium plates moduļus var iedalīt divās daļās: moduļi, kuri darbojas kamēr platei tiek pievadīta strāva (piem. USB) un moduļi, kuri darbojas, kad tiem barošana tiek pieslēgta no mikrokontroliera programmas koda (piem. RTC un akselerometrs). Abu moduļu veidus inicializē un lieto atšķirīgos veidos:

- USB interfeisa pielietojumi

```
{
  USB.begin(); // Inicializē UART - USB interfeisu
  USB.print("teksts"); //nosūta caur UART kādus datus
  USB.println(); //nosūta caur UART kādus datus un beigās pievieno EOL (end of line)
  USB.available(); // atgriež 1, ja buferī atrodas dati
  uint8_t data_read=USB.read(); // nolasa bufera datus
  USB.flush(); // atbrīvo UART buferi
  USB.close(); // aizver komunikāciju enerģijas taupīšanai
}
```

- RTC inicializācija un pielietošana

```
{
  RTC.ON();//pieslēdz RTC pie barošanas avota
  RTC.setTime("01:01:01:01:01:01");//uzstāda RTC datumu un laiku
  RTC.getTime();//atgriež datumu un laiku
  RTC.getTemperature();//atgriež RTC iebūvētā temperatūras sensora vērtību °C
}
```

Laboratorijas darba gaitā ir jāveic divu moduļu inicializācija (USB un RTC). UART pēc noklusējuma tiek parametri ir: baudrate=38400, 8 datu biti un 1 stop bits. Waspnote moduļa programmēšana var notikt tad, kad nav atvērta UART-USB komunikācija ar datoru.

RTC tiek barots divos veidos, no ārējā barošanas avota un no RTC baterijas, kas ļauj saglabāt ieprogrammēto RTC datumu un laiku, kad ārējais barošanas avots tiek atslēgts, vai waspmote plate tiek uzstādīta enerģiju taupošajā režīmā. Datums un laiks RTC tiek uzstādīts speciālā formā, 'YY:MM:DD:dow:hh:mm:ss' respektīvi gads, mēnesis, datums, diena nedēļā, stunda, minūte, sekunde. Šī formāta īpatnība ir diena nedēļā, kas ir skaitlis no 1 līdz 7, attiecīgi norādot 1 - svētdiena, 2 - pirmdiena, ..., 7 - sestdiena. RTC modulī ir iebūvēts arī temperatūras sensors, kurš dos precīzāku

informāciju tieši par RTC moduļa temperatūru, nevis par apkārtējās
vides temperatūru.

Darba uzdevumi:

- Noteikt akselerometra kļūdas;
- Noteikt kāpēc tādas rodas;
- * Piedāvāt risinājumu dažādu sensoru mērījumu precizitātes uzlabošanai.

Apraksts:

Laboratorijas darbā ir jāiepazīstas ar vienu no waspmote sensoriem, akselerometru. Akselerometrs ir sensors, kurš mēra paātrinājuma spēku, jeb vienkārši, paātrinājumu. Tie ir dažādi pēc mērāmo dimensiju skaita un jūtības. Iebūvētais akselerometrs mēra paātrinājumu pa visām trim asīm (X, Y un Z asīm). Šie mērījumi var būt gan statiski, gan dinamiski atkarībā no akselerometra mērāmās vides. Dinamiskie akselerometra mērījumi ļauj noteikt kādas ierīces pārvietošanos, pie kuras ir piestiprināts akselerometrs. Paātrinājuma mērīšana ir nepieciešama ļoti reti, tomēr apvienojot akselerometru ar kādu citu sensoru, var iegūt daudz noderīgākus mērījumus.

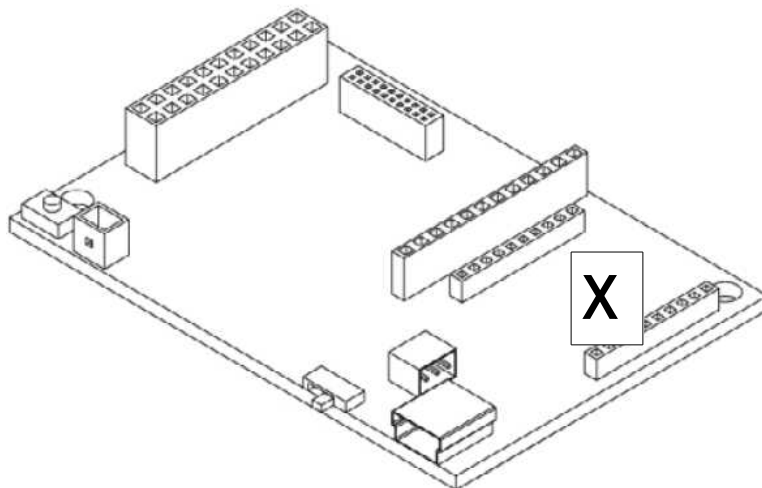
Dotajā darbā akselerometrs tiks apvienots kopā ar taimeri/skaitītāju, lai veiktu aptuvenu distances mērīšanu. No fizikas viedokļa pārvietojums ir lielums, kas raksturo ātruma izmaiņas laikā un tā SI mērvienība ir m/s^2 . Tātad ņemot akselerometra izmērīto paātrinājumu un taimera/skaitītāja doto laiku, iespējams aprēķināt kustības ātrumu pēc formulas $v = a \cdot t$, kur v – ātrums un t – laiks. Ja ir zināms pārvietošanās ātrums un laiks, tad ir iespējams arī aprēķināt distanci no sākumpunkta līdz beigu punktam.

Ražotājs ir pacenties unapgādājis waspmote plates lietotājus ar visām nepieciešamajām bibliotēkām, kuras atvieglo programēšanu darbā ar dažādām papildierīcēm. Dotajā gadījumā atliek inicializēt akselerometru un taimeri/skaitītāju.

```
{
    ACC.ON(); // Initializes the accelerometer
    uint8_t status=ACC.getStatus(); // Gets the accelerometer status
    status=ACC.check(); // Gets if accelerometer is present and is working properly
    int16_t accX=ACC.getX(); // Gets acceleration on X axis.
    int16_t accY=ACC.getY(); // Gets acceleration on Y axis.
    int16_t accZ=ACC.getZ(); // Gets acceleration on Z axis.
    int16_t flag=ACC.setFF(); // Sets Free Fall interruption
    flag=ACC.unsetFF(); // Unsets Free Fall interruption
    flag=ACC.setDD(); // Sets Direction Change interruption with pre-defined direction
    uint8_t direction=YHIE | YLIE;
    flag=ACC.setDD(direction); // Sets Direction Change interruption
    flag=ACC.unsetDD(); // Unsets Direction Change interruption
    ACC.setMode(ACC_HIBERNATE); // Sets ACC to Hibernate Mode
    uint8_t workMode=ACC.getMode(); // Gets ACC Work Mode
    ACC.setAccEvent(ACC_FREE_FALL); // Sets Event to Free Fall
    uint8_t eventType=ACC.getAccEvent(); // Gets Event
    uint8_t adcMode=ACC.getADCmode(); // Gets the ADC mode: 12 or 16 bits
    uint8_t adcMode=1; ACC.setADCmode(adcMode); // Sets ADC Mode to 16 bit left justified
    ACC.setSamplingRate(ACC_RATE_640); // Sets a 640Hz sampling rate
    ACC.close(); // Closes I2C bus and sets accelerometer to hibernate mode
}
```

Lai būtu iespējams noteikt sensora kļūdu, ir nepieciešams veikt vairākus pilnīgi vienādus testus, kuru rezultāti beigās tiek salīdzināti. Dotajā laboratorijas darbā nav iespējams veikt tiešus akselerometra testus, jo nav pieejama nepieciešamā aparatūra.

Izmantojot iepriekš aprakstītās akselerometra komandas un mikrokontroliera taimeri/skaitītāju nepieciešams aprēķināt kāda neliela pārvietojuma ātrumu un distanci. Kamēr waspmote plate ar visu akselerometru atrodas kustībā, tai pastāv kaut kād paātrinājums, kas attiecīgi tiek izmērīts ar akselerometru. Savukārt, kad plate atrodas miera stāvoklī, paātrinājums ir vienāds



ar 0. Izmantojot taimeri/skaitītāju nepieciešams izmērīt laiku, kamēr plate atrodas kustībā.

Pašas waspmote plates akselerometra mērāmās dimensijas ir attēlotas attēls II.. Dotais **Z**

Y

X

akselerometrs satur 12-bitu ADC un tam ir divi mērīšanas režīmi $\pm 2g$ un $\pm 6g$.

Jautājumi atskaitei:

- Cik reižu sekundē akselerometrs ir spējīgs veikt mērījumus un kurā akselerometra reģistrā šī vērtība tiek mainīta?
- Kāpēc ir atšķirība starp reālo distanci un ar plati izmērīto?
- * Kādas metodes pastāv paātrinājuma mērīšanai?

laboratorijas darbs. Enerģiju taupošais režīms.

Darba uzdevumi:

- Ieslēgt/izslēgt enerģiju taupošo režīmu;
- Likst platei iziet no enerģiju taupošā režīma sakarā ar kādu notikumu (pārtraukumu);
- * Pārbaudīt waspmote plates deepSleep un hibernate režīmus.

Apraksts:

Wasmote platei ir iespējami vairāki enerģiju taupošie režīmi, kurus lieto dažādās situācijās. Visbiežāk tiek lietots *sleep* režīms, kas ļauj ietaupīt akumulatora enerģiju. Pielietojot dažādus sensorus un papildus moduļus, enerģijas patēriņš pieaug. Tieši tāpēc ražotāji ir pacentušies un izveidojuši iespēju no programmas koda ieslēgt un izslēgt dažādus moduļus, lai tie bez vajadzības netērētu enerģiju. Gadījumos, kad akumulatora enerģija tuvojas beigām, tiek lietots *hibernate* stāvoklis, kurā waspmote plate saglabā patreizējo plates stāvokli. Protams no šiem visiem enerģiju taupošajiem stāvokļiem kādreiz ir arī jāiziet un jāturpina programmas izpilde.

Iespējamie waspmote plates režīmi:

- **ON**: waspmote plate darbojas pilnībā un patērē ap **9mA**;
- **SLEEP**: mikrokontroliera programmas izpildīšana tiek apstādināta. Šajā režīmā ir iespējams saņemt pārtraukumus no sensoriem un taimeriem. Režīms tiek kontrolēts ar *Watchdog* taimerī. Wasmote plate šajā režīmā var atrasties no 32ms līdz 8s un patērēt aptuveni **62μA**;
- **DEEP SLEEP**: mikrokontroliera programmas izpildīšana tiek apstādināta. Šajā režīmā ir iespējams saņemt pārtraukumus no sensoriem un taimeriem. Režīms tiek kontrolēts ar RTC taimerī. Wasmote plate šajā režīmā var atrasties sākot no 8s līdz vairākām minūtēm, stundām, dienām vai pat gadiem un patērēt aptuveni **62μA**;
- **HIBERNATE**: pats mikrokontrolieris un visi plates moduļi ir pilnībā izslēgti. Šajā režīmā ir iespējams saņemt tikai RTC modinātāja signālus, pēc kura plate tiek iedarbināta. Pēc darbības atsākšanas plate nav atiestatīta (nav noticis RESET), kas vienkārši ļauj turpināt izpildīt mikrokontroliera programmu no tās vietas, kur tā tika apturēta. Šajā režīmā waspmote plate var atrasties sākot no 8s līdz vairākiem gadiem un patērēt niecīgu daudzumu enerģijas, aptuveni **0.7μA**.

Strāva Plate

	as patēr inš	s stāvo klis	Cikla ilgums	Saņemamie pārtraukumi
ON	9mA	Ieslē gta	-	Visi iespējamie
SLEEP	62μA	Ieslē gta	32ms - 8s	Sensori un Watchdog taimeris
DEEP	62μA	Ieslē	8s -	Sensori un RTC

SLEEP		gta	minūtes/stundas/dienas/ gadi	
HIBERNAT E	0.7μA	Izslē gta	8s - minūtes/stundas/dienas/ gadi	RTC

Platei ir arī vairāki enerģijas taupīšanas parametri, pēc kuriem arī tiek atslēgti konkrētie moduļi:

- ALL_OFF: atslēdz pilnīgi visus waspmote platē iestrādātos slēdžus, ieskaitot UART komunikāciju;
- SENS_OFF: pilnībā atslēdz waspmote platei pievienotās sensoru plates;
- UART0_OFF: aizver UART0 komunikāciju un iestrādāto slēdzi priekš Xbee komunikācijas;
- UART1_OFF: aizver UART1 komunikāciju un iestrādāto slēdzi priekš GPS, GPRS vai *Expansion Board*;
- BAT_OFF: atslēdz akumulatora enerģijas līmeņa mērīšanu;
- RTC_OFF: atslēdz RTC no pamat-barošanas avota.

Visus waspmote enerģiju taupošos režīmus var programmēt izmantojot jau ražotāja piedāvātās bibliotēkas, kurās ir ietverti visi uzskaitītie režīmi un to parametri. Enerģiju taupīšanas komandas ir sekojošas:

```
{
  PWR.sleep(UART0_OFF | BAT_OFF);
  // miega režīms atslēdzot UART0 interfeisu un baterijas monitoringu
  PWR.sleep(WTD_32MS, ALL_OFF);
  // miega režīms atslēdzot visus moduļus uz 32milisekundēm
  PWR.deepSleep("00:00:00:10", RTC_OFFSET, RTC_ALM1_MODE2, ALL_OFF);
  // miega režīms atslēdzot visus moduļus uz 10 sekundēm
  PWR.hibernate("15:17:00:00", RTC_ABSOLUTE, RTC_ALM1_MODE2, ALL_OFF);
  // miega režīms līdz mēneša 15-ajam datumam plkst 17.00
  PWR.ifHibernate();
  // Checks if we come from a normal reset or an hibernate reset
  PWR.setSensorPower(SENS_5V, SENS_OFF);
  // Sets the 5V switch OFF
  PWR.setWatchdog(WTD_ON, WTD_2S);
  // Enables Watchdog to generate interruption after 2 seconds
  PWR.clearInts;
  // Clears interruptions on modules that have generated one
  uint8_t battery_level = PWR.getBatteryLevel();
  // battery_level contains the % of remaining battery
  PWR.setLowBatteryThreshold();
  // sets the low battery threshold
}
```

Jautājumi atskaitei

- Kādos gadījumos var izmantot katru no enerģiju taupošajiem režīmiem?
- Vai mikrokontrolieris ir spējīgs strādāt paralēli atrodoties *SLEEP* režīmā? Ja jā, tad paskaidrot kā un kāpēc?
- * Kā notiek un kā izpaužas *HIBERNATE* režīma ieslēgšana un izslēgšana uz mikrokontrolieriem?

laboratorijas darbs. Sensoru plates sensoru mērījumu nolasīšana.

Darba uzdevumi:

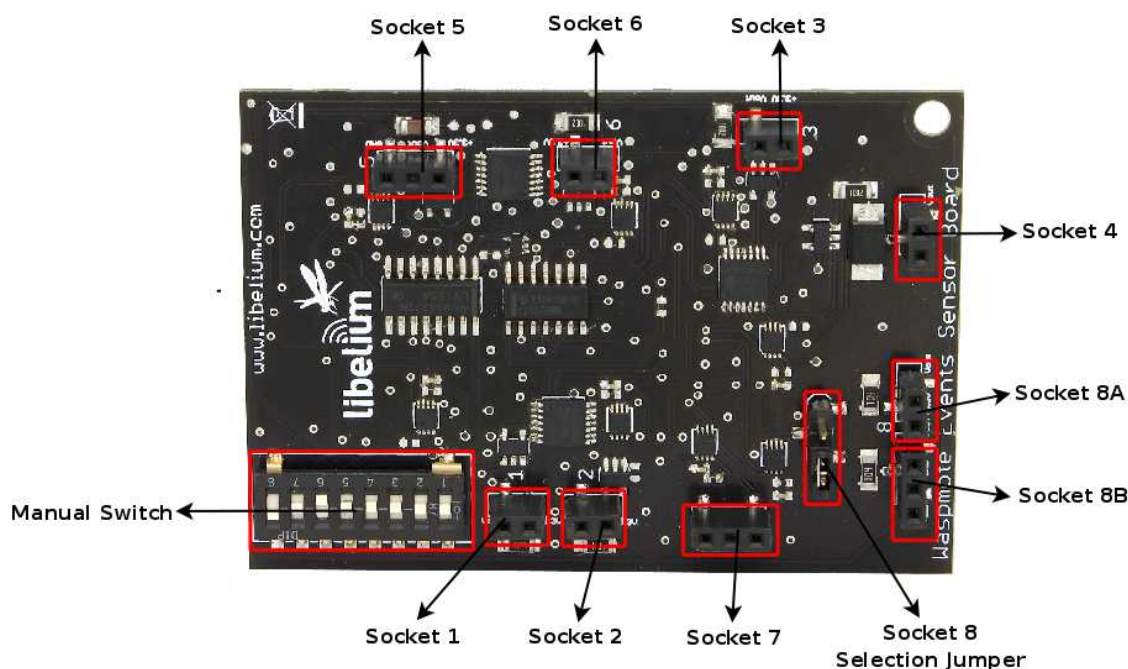
- Darbam ar *Events Sensor* plati:
 - Iegūt PIR, temperatūras, gaismas intensitātes, vibrācijas, un citu sensora datus;
 - Atkarībā no PIR sensora datiem veikt pārējo sensoru mērījumu nolasīšanu un izvadīšanu;
- Darbam ar *Gas Sensor* plati:
 - Katru minūti veikt iekštelpas gaisa sastāva mērījumus;
 - Atbilstoši temperatūras sensora izmaiņām veikt visu pārējo sensoru mērījumu nolasīšanu un izvadīšanu;
- * Pieslēgt pie waspmote plates kādu no laboratorijā pieejamajiem sensoriem, kurš izmanto kādu no interfeisiem (UART vai I²C [**SPI interfeiss nav pieejams**]) un izvadīt tā mērījumus.

Apraksts:

Laboratorijas darba izpildīšanai ir pieejamas divas veidu sensoru plates, *Events sensor board* – dažādu notikumu noteikšanai un *Gases sensor board* – gāzu sastāva noteikšanai. Sensori katrai platei ir dažādi, izņemot temperatūras sensoru, kuru var lietot uz abām platēm. Lielākā sensoru izvēle laboratorijā ir tieši priekš *Events sensor* plates – vairāk kā 7 dažādi sensori, bet priekš *Gases sensor* plates – tikai 6 sensori. Pašas plates ar pieslēgtiem sensoriem tiks dotas atsevišķi. **No plates atvienot vai pievienot kādu citu sensoru drīkst tikai ar pasniedzēja atļauju.**

Events Sensor board

Notikumu sensnsoru platei ir iespējams daudz dažādu sensoru, kuru izejās ir vai nu analogs signāls vai arī digitāls signāls. Analogais signāls, piemēram, temperatūras sensora mērījumi, tiek pārveidoti izmantojot mikrokontroliera ADC, savukārt digitālo



signālu atliek vienkārši nolasīt, jo tam ir tikai divas dažādas vērtības.

Dotā plate vienlaicīgi spēj strādāt ar 8 sensoriem, kuru izejas tiek salīdzinātas ar kādu uzstādītu sliekšņa vērtību un tiek sensoru signāli caur loģisko OR mikroshēmu tiek kombinēti, tādā veidā ļaujot mikrokontrolierim reaģēt uz astoņiem dažādiem pārtraukumiem. Šīs sliekšņa vērtības tiek uzstādītas ar digitālo potenciometru palīdzību [lit-avots].

Pieejamie sensori priekš šīs plates ir sekojoši:

- ⊗ Vibrācijas sensors PZ-08;
- ⊗ Vibrācijas sensors AG2401-1;
- ⊗ Triecienu sensors ASLS-2;
- ⊗ Leņķa sensors AG3011-1;
- ⊗ Temperatūras sensors MCP9700A;
- ⊗ Luminiscences sensors;
- ⊗ Kustību sensors PIR.

Programmēšanā tiek izmantotas ražotāja dotās bibliotēkas, kurās jau ir realizētas visas nepieciešamās funkcijas sensoru mērījumu nolasīšanai. Ir jābūt uzmanīgiem pievienojot klāt citus sensorus, jo katra pieslēgvietā ir veidota dažādi, tāpēc ne visās pieslēgvietās

attēls II. Events board

būs iespējams pievienot kādu no sensoriem. Ja tiek ņemts vērā, kur un kāds sensors ir jāpieslēdz, tad vērtību nolasīšanai pietiek ar vienu komandu pilnīgi visām pieslēgvietām. Dotās plates komandas ir šādas:

```
{
  // aktivizē plati un pislēdz sensoru plati barošanas avotam
```



```

SensorEvent.setBoardMode(SENS_ON);
// uzstāda sliekšni ar vērtību 1.5V uz SOCKET1 pieslēgvietas
SensorEvent.setThreshold(SENS_SOCKET1, 1.5);
// atļauj platei ģenerēt pārtraukumus
SensorEvent.attachInt();
// aizliedz platei ģenerēt pārtraukumus
SensorEvent.detachInt();
// nolasa pārtraukumu ģenerējošās sensoru pieslēgvietas
SensorEvent.loadInt();
// saglabā iepriekšējās komandas rezultātu
uint8_t flag = SensorEvent.intFlag;
// nolasa sensora vērtību pieslēgtu pie SOCKET1 pieslēgvietas. Ja vērtība ir analogā tad
// tiek atgriezta sprieguma vērtība, bet ja digitālais - tad '0' vai '1'.
uint8_t value = SensorEvent.readValue(SENS_SOCKET1);
}

```

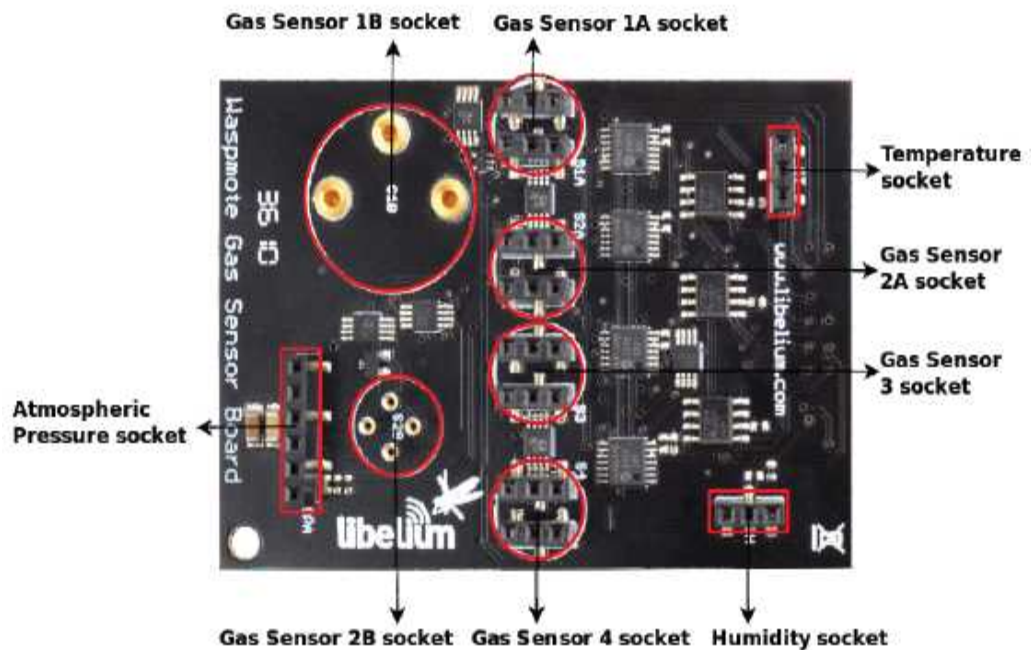
Gas Sensor board

Gāzu sensoru platei ierobežots skaits sensoru, kas ir pieejami laboratorijā. Ar šīs plates palīdzību var noteikt gaisa un citu gāzu sastāvu, temperatūru, spiedienu un arī gāzes/gaisa mitrumu (ūdens daudzumu). Kopumā ar šo plati ir iespējams noteikt 14 dažādas gāzes, bet pie pašas plates vienlaicīgi var būt pieslēgti tikai 6 gāzu sensori. Uz plates ir unikālas pieslēgvietas, pie kurām var pieslēgt tikai konkrētus sensorus: temperatūras, mitruma, spiediena, CO₂ un O₂ sensorus.

Ar plati var izmērīt sekojošas ķīmiskas gāzes:

- | | |
|---|---|
| ⊗ | |
| ⊗ Oglekļa monoksīds (CO); | ⊗ Etanols (CH ₃ CH ₂ OH); |
| ⊗ Oglekļa dioksīds (CO ₂); | ⊗ Tuluols (C ₆ H ₅ CH ₃); |
| ⊗ Skābeklis (O ₂); | ⊗ sērūdeņražskābe (H ₂ S); |
| ⊗ Metāns (CH ₄); | ⊗ Slāpekļa dioksīds (NO ₂); |
| ⊗ Ūdeņradis (H ₂); | ⊗ Ozons (O ₃); |
| ⊗ Amonjaks (NH ₃); | ⊗ Gaistošos organiskos savienojumus un ogļhidrātus |
| ⊗ Butāns (C ₄ H ₁₀); | |

Attēlā var redzēt gāzu sensora plates iespējamās pieslēgvietas un to numurus. Pašas plates aprakstā [**litavots**] var atrast pašus sensorus, kur kādus sensorus var pieslēgt un kādi ir katra sensora parametri. Tā kā katrs sensors mēra dažādas gāzes, tad šo sensoru mērījumu nolasīšanai ir izveidota viena funkcija, tikai katram sensoram ir savi funkcijas parametri. Šos parametrus arī var atrast gāzu sensoru plates aprakstā.



Pamata funkcijas darbībai ar plati:

```
{
    // aktivizē plati un pieslēdz sensoru plati barošanas avotam
    SensorGas.setBoardMode(SENS_ON);
    // aktivizē norādīto sensoru atsevišķi
    SensorGas.setSensorMode(SENS_ON, SENS_PRESSURE);
    // uzstāda dažādus parametrus, kuri ietekmēs sensora mērījumus
    SensorGas.configureSensor(SENSOR, GAIN, RESISTOR);
    // nolasa mitruma sensora vērtību.
    uint8_t value = SensorGas.readValue(SENS_HUMIDITY);
}
```

Jautājumi atskaitei:

- Noteikt, ko un kā mēra katrs no laboratorijas darbā izmantotajiem sensoriem?
- Piedāvāt vismaz trīs pielietojumus izmantotajiem sensoriem?
- * Ar kādiem sensoriem un kā ir iespējams izmērīt attālumu līdz kādam objektam? Kas ierobežo mērīšanas distanci?

laboratorijas darbs. Xbee paketes veidošana un sūtīšana/saņemšana.

Darba uzdevumi:

- Izveidot XBee paketi un nosūtīt to uz citu waspmote plati, izmantojot:
 - 64-bitu MAC adresi;
 - 16-bitu MAC adresi;
 - Mezgla identifikatoru;
- Caur vārtejas moduli, kurš pieslēgts pie datora, saņemt izveidoto Xbee paketi;
- * Izmantojot Xbee moduļa digitālās ieejas/izejas (pieejamas uz vārtejas moduļa), nosūtīt waspmote platei pieprasījumu pēc datu paketes.

Apraksts:

Dotajā laboratorijas darbā ir jāizmanto XBee bezvadu tīkla raidītājs/uztvērējs redzams attēls II.. Šo moduli uz waspmote plates var pieslēgt pie SOCKET0. Error: Reference source not found ir parādītas abas iespējamās bezvadu moduļu pieslēgvietas uz waspmote plates. SOCKET1 pieslēgvietā ir pieejama tikai caur speciālu papildmoduli. Laboratorijas darbā izmantojamais Xbee modulis strādā no 2.405 – 2.465GHz frekvences diapazonā. Datu pārraides jauda sasniedz 18dBm(63mW), bet jūtība sasniedz 100dBm. Modulis atbalsta IEEE 802.15.4 standartu, kurš definē fiziskālā un tīkla slāņa parametrus. Vienkāršas komunikācijas izveidošanai pietiek P2P (Peer-to-Peer) kurā sazinās tikai divas plates.



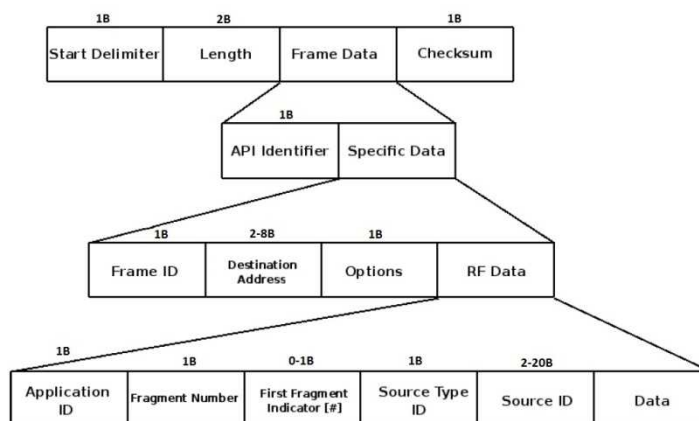
attēls II. Xbee modulis

pielietot
topoloģiju,
waspote

et 1

Datu sūtīšana

Katra bezvadu tīklā nosūtītā pakete satur tās galveni ar dažādiem tās parametriem un norādītām mērķa un avota adresēm. Paketē mērķa adrese var būt norādīta vai 16-bitu vai 64-bitu garumā. Unikālā 64-bitu MAC adrese ir ražotāja piešķirta un to var nolasīt ar speciālu komandu, bet 16-bitu adresi ir jāpiešķir manuāli. Kā var redzēt **attēlā**, tad pirms datiem tiek norādīti vēl papildus parametri, kā paketes garums, paketes numurs un citi. Daudzi no šiem parametriem var tikt uzstādīti automātiski, tomēr saņēmēja adreses būs jāuzstāda pašiem.



attēls II. Paketes struktūra

IEEE 802.15.4 atbalsta divus no trim pakešu pārraides veidiem: BROADCAST un UNICAST.

- ⊗ UNICAST datu pārraidē, pēc noklusējuma, atrodas visi inicializētie waspmote plates Xbee moduļi un tā atbalsta apstiprinājuma (ACK) pakešu saņemšanu. Gadījumā ja ACK pakete netiek saņemta, sūtītājs atkārtoti nosūtīs iepriekšējo paketi saņēmējam līdz saņēmējs atsūtīs atpakaļ ACK paketi. Šajā režīmā ir iespējamas abu veidu adreses:
 - **16-bitu adrese.** Xbee moduļus ir iespējams nokonfigurēt, lai būtu iespējams lietot 16-bitu adreses. To ir iespējams paveikt norādot paketes parametrus 16-bitu adresi un adrese veidu, konkrēti MY_TYPE. Abas waspmote plates var komunicēt savā starpā izmantojot 16-bitu adreses tādā gadījumā, ja mērķa adrese datu sūtītāja platē sakrīt ar datu saņēmēja adresi.

	WASPMOTE_1	WASPMOTE_2
Avota adrese	0x01	0x02
Mērķa adrese	0x02	0x01

 - **64-bitu adrese.** Katram Xbee modulim ir ierakstīta unikāls 64-bitu kods, kuru var izmantot kā MAC adresi. Šis adreses veids tiek lietots tādos gadījumos, kad tiek norādīts MAC_TYPE adreses veids.

- ⊗ BROADCAST datu pārraidē apstiprinājuma pakete atpakaļ netiek sūtīta un arī automātiska atkārtota paketes

nosūtīšana nav iespējama. Šādā pārraides veidā pilnīgi visi tīklā esošie Xbee moduļi saņems paketes, kuras saturēs apraides adresi. Šis pārraides veids tiek izmantots lai veidotu savienojumu ar kādu mezglu, uzzinātu tā parametrus un citus datus.

Datu sūtīšanas uzsākšanai ir nepieciešams konfigurēt abus Xbee moduļus datu sūtīšanai un saņemšanai. Sākumā ir jāizvēlas kāds no 12 kanāliem, pa kuru notiks šī komunikācija, vienkārši norādot atbilstošo HEX vērtību no tabulas[**atsauce uz tabulu ar kanālu numuriem HEX un frekvenci**]. Tāpat jāveic waspmote plates adreses veida un pašas adreses izvēle. Iepriekš tika norādītas 16 un 64-bitu adreses, bet papildus iespējams arī norādīt kādu līdz 20 simboliem garu kodu. Kods var būt arī kā nosaukums, kuram jābūt unikālam un tam obligāti ir jābeidzas ar simbolu '#', tad šis kods tiks pieņemts kā mērķa vai avota adrese.

Nr. p.k	Kanāla numurs	Darbības frekvences diapazons
1	0x0C - kanāls nr. 12	2.405 - 2.410 GHz
2	0x0D - kanāls nr. 13	2.410 - 2.415 GHz
3	0x0E - kanāls nr. 14	2.415 - 2.420 GHz
4	0x0F - kanāls nr. 15	2.420 - 2.425 GHz
5	0x10 - kanāls nr. 16	2.425 - 2.430 GHz
6	0x11 - kanāls nr. 17	2.430 - 2.435 GHz
7	0x12 - kanāls nr. 18	2.435 - 2.440 GHz
8	0x13 - kanāls nr. 19	2.440 - 2.445 GHz
9	0x14 - kanāls nr. 20	2.445 - 2.450 GHz
10	0x15 - kanāls nr. 21	2.450 - 2.455 GHz
11	0x16 - kanāls nr. 22	2.455 - 2.460 GHz

0x17 -
12 kanāls nr. 2.460 - 2.465 GHz
23

Programmēšana:

Lai būtu pēc iespējas vienkāršāk konfigurēt pašu Xbee moduli, Libelium ražotāji ir izveidojuši speciālas bibliotēkas šo moduļu konfigurēšanai papildus pievienojot datu sūtīšanas un saņemšanas iespējas. Pēc noklusējuma visi Xbee bezvadu komunikāciju moduļi strādā kanālā 0x0C. Kā jau ar visiem iepriekš strādātiem moduļiem, arī šo pašā sākumā ir nepieciešams inicializēt un pieslēgt pie barošanas avota. Pēc šī soļa var sākties paša moduļa parametru uzstādīšana un tikai tad būs iespējams sūtīt datus.

Praktiskās daļas realizēšana notiks izmantojot divus Xbee moduļus, kur viens būs pieslēgts pie waspmote plates, bet otrs - uz vārtejas plates. Vārtejas plates Xbee moduli būs nepieciešams konfigurēt tā, lai tas strādātu tajā pašā tīklā, kurā strādās waspmote plate. Tīkla numurs tiek dots katram atsevišķi no pasniedzēja. Lai būtu iespējams konfigurēt vārtejas plates Xbee moduli, būs jāprogrammē visi iestatījumi caur waspmote plati. Sākumā pie waspmote plates būs jāpieslēdz vārtejas Xbee modulis un jāieraksta tajā visi tīkla uzstādījumi un tikai tad būs iespējams realizēt un pārbaudīt komunikāciju. Izmantojot *writeValues()* komandu, Xbee modulis saglabāt visus iestatījumus arī pēc tā izslēgšanas. Visu nepieciešamo parametru uzstādīšana un datu sūtīšana tiek realizēta ar šīm komandām:

```
{
xbee802.init(XBEE_802_15_4,FREQ2_4G,NORMAL); // inicializē XBee moduli
xbee802.ON(); // ieslēdz XBee moduli
xbee802.OFF(); // izslēdz XBee moduli
xbee802.encryptionMode(1); // ieslēdz tīkla aizsardzību
xbee802.setLinkKey(key); // uzstāda tīkla drošības kodu
xbee802.setChannel(0x0D); // uzstāda kanālu
xbee802.getChannel(); // ---
xbee802.setMacMode(2); // uzstāda Mac režīmu 802.15.4 galvenē
xbee802.getMacMode(); // ---
xbee802.getOwnMacLow(); // nolasa 32 LSB bitus no MAC adreses
xbee802.getOwnMacHigh(); // nolasa 32 MSB bitus no MAC adreses
xbee802.setOwnNetAddress(0x12,0x34); // uzstāda 0x1234 kā moduļa adresi
xbee802.getOwnNetAddress(); // ---
xbee802.setPAN(papid); // uzstāda PANID
xbee802.getPAN(); //---
xbee802.setNodeIdentifier("forrestnode-01#"); // uzstāda 'forrestnode-01' kā plates adresi
xbee802.getNodeIdentifier(); // ---
xbee802.setScanningChannels(0xFF,0xFF); // skenēt pilnīgi visus kanālus
xbee802.getScanningChannels(); // saņem kanālu sarakstu
xbee802.getACKcounter(); // nolasi ACK skaitu
xbee802.resetACKcounter (); // inicializēt ACK skaitītāju
xbee802.setRetries(2); // uzstādīt pakešu atkārtotas sūtīšanas skaitu
xbee802.getRetries(); // ---
xbee802.setDurationEnergyChannels(3); // uzsākt enerģijas skenēšanu pa kanāliem
xbee802.writeValues(); // saglabāt uzstādījumus pēc plates izslēgšanas
xbee802.sendCommandAT("CH#"); // nosūtīt "ATCH" komandu XBee modulim
xbee802.getRSSI(); // nolasi Receive Signal Strength Indicator
xbee802.send("0013A2004030F66A",data); // nosūtīt modulim datus
xbee802.setOriginParams(paq_sent, "1221", MY_TYPE); // uzstādīt sūtītāja parametrus
xbee802.setDestinationParams(paq_sent, "1234", data, MY_TYPE, DATA_ABSOLUTE); // noteikt saņēmēja parametrus
state=xbee802.sendXBee(paq_sent); // nosūtīt datus tīklā
}
```

Šāda veida pakete ir strukturēta API bibliotēkās un definēta kā „packetXBee”. Struktūrai ir daudz mainīgo un tie ir sagrupēti trijās

grupās, iekšējie, aplikācijas un ārējie mainīgie. Iekšējie mainīgie paredzēti paketes parametru un sūtīšanas parametru uzstādīšanai, piemēram, kāda veida pakete tiek sūtīta. Aplikācijas mainīgie norāda paketē norādītās mērķa adreses veidu un sūtīšanā izmantotā maršruta adreses, savukārt ārējie mainīgie norāda paketes stāvokli, piemēram, vai pakete ir piegādāta vai arī cik reizes sūtīšana tiek atkārtota, līdz pakete tiek saņemta.

```
/****** IEKŠĒJIE *****/
uint8_t macDL[4];      // 32b Lower Mac Destination
uint8_t macDH[4];      // 32b Higher Mac Destination
uint8_t mode;          // 0=unicast ; 1=broadcast ; 2=cluster ; 3=synchronization
uint8_t address_type;  // 0=16B ; 1=64B
uint8_t naD[2];        // 16b Network Address Destination
char data[MAX_DATA];   // Data of the sent message. All the data here, even when > Payload
uint16_t data_length;  // Data sent length. Real used size of vector data[MAX_DATA]
uint16_t frag_length;  // Fragment length. Used to send each fragment of a big packet
uint8_t SD;            // Source Endpoint
uint8_t DE;            // Destination Endpoint
uint8_t CID[2];        // Cluster Identifier
uint8_t PID[2];        // Profile Identifier
uint8_t MY_known;      // 0=unknown net address; 1=known net address
uint8_t opt;           // options: 0x08=Multicast transmission

/****** APLIKĀCIJAS *****/
uint8_t packetID;      // ID for the packet
uint8_t macSL[4];      // 32b Lower Mac Source
uint8_t macSH[4];      // 32b Higher Mac Source
uint8_t naS[2];        // 16b Network Address Source
uint8_t macOL[4];      // 32b Lower Mac Origin Source
uint8_t macOH[4];      // 32b Higher Mac Origin Source
uint8_t naO[2];        // 16b Network Address origin
char niO[20];          // Node Identifier Origin. To use in transmission, it must finish "#".
uint8_t RSSI;          // Receive Signal Strength Indicator
uint8_t address_typeS; // 0=16B ; 1=64B
uint8_t typeSourceID;  // 0=naS ; 1=macSource ; 2=NI
uint8_t numFragment;   // Number of fragment to order the global packet
uint8_t endFragment;   // Specifies if this fragment is the last fragment of a global packet
uint8_t time;          // Specifies the time when the first fragment was received

/****** ĀRĒJIE *****/
uint8_t deliv_status;  // Delivery Status
uint8_t discov_status; // Discovery Status
uint8_t true_naD[2];   // Network Address the packet has been really sent to
uint8_t retries;       // Retries needed to send the packet
```

Tā kā sūtīšana notiek starp diviem mezgliem, tad pilnīgi visus iepriekš minētos paketes struktūras mainīgos nemaz nevajag izmantot. Pietiek norādīt mērķa, avota un tīkla adreses, sūtīšanas režīmu, adreses veidu un ēl dažus parametrus atkarībā no pielietojuma.

Jautājumi atskaitei:

- Kur var tikt pielietots katrs no paketes struktūras mainīgajiem?
 - Kas ir signāla stiprums un kā to var (ja var) ietekmēt?
 - * Vai ir iespējams attālināti kontrolēt Xbee moduļa digitālās ieejas/izejas? (ja ir, tad kā to var paveikt?)

laboratorijas darbs. XBee 802.15.4 tīkla veidošana.

Darba uzdevumi:

- Kopīgi izeidot 802.15.4 bezvadu sensoru tīklu:
 - 1 koordinators;
 - 2 maršrutētāji;
 - 4 gala sistēmas datu vākšanai;
- Izmantojot enerģiju taupošos režīmus, pēc iespējas efektīgāk izmantot akumulatoru enerģiju;
- Veikt dažādu datu vākšanu un nosūtīšanu līdz datoram;
- * Piedāvāt risinājumus energoefektivitātes uzlabošanai laboratorijas darbā izveidotajam tīklam.

Darba veikšanai studentiem kopīgi jāizlemj par to, kurš veidos konkrēto, uzdevumos norādīto tīkla elementu. Ja tas nav iespējams, tad pasniedzējs var likt katram studentam veidot un konfigurēt atsevišķu tīkla elementu.

Apraksts:

Laboratorijas darbs tiek balstīts uz iepriekšējo laboratorijas darbu, tikai šajā darbā ir jāizveido tīkls speciāli konfigurējot katru waspmote plati. Tīkla koordinators ir elements, kurš veic tīkla inicializāciju, sūta pieprasījumus pēc datiem un veic citas darbības tīklā. Pats koordinators izveido atsevišķu tīklu, pie kura var pieslēgties gan maršrutētāji, gan gala sistēmas. Maršrutētājs attiecīgi veic pakešu tālāku nosūtīšanu no koordinatora līdz gala sistēmai dažādos tīklos. Tātad arī maršrutētājs izveido tīklu, pie kura pieslēdzas citi maršrutētāji un gala sistēmas, kā arī pats maršrutētājs pieslēdzas pie koordinatora tīkla, lai būtu iespējams nosūtīt gala sistēmu sūtītās paketes tālāk uz koordinatoru. Gala sistēmas var tikai pieslēgties klāt pie kāda no izveidotajiem tīkliem un veikt datu vākšanu un nosūtīšanu. Visi tīkla elementi satur informāciju par tīkla koordinatoru un maršrutētājiem, pie kuriem konkrētais tīkla elements ir pieslēdzies. Paša tīkla darbības soļi ir sekojoši:

- 1) Koordinators veic tīkla inicializāciju un izvēlas tīkla adresi un kanālu;
- 2) Maršrutētājs skenējot visus kanālos, sameklē koordinatora veidoto tīklu un pieslēdzas pie tā;
- 3) Maršrutētājs saglabā koordinatora tīkla iestatījumus(tīkla adresi un koordinatora adresi) un inicializē jaunu tīklu, pie kura pieslēgsies gala sistēmas;
- 4) Tāpat kā maršrutētāji, tāpat arī gala sistēmas meklē maršrutētāja veidoto tīklu, pie kura pieslēgties;
- 5) Kad gala sistēma ir pieslēgusies pie maršrutētāja, tad gala sistēmām tiek nosūtīta informācija par koordinatoru, kurā tīklā tas atrodas un kāda ir tā adrese.

Jautājumi atskaitē:

- Kā darbojas katrs no laboratorijas darbā konfigurētajiem (programmas kodā) moduļiem bezvadu sensoru tīklā?

- Kā notiek datu sūtīšana no gala sistēmas līdz koordinatoram un datoram?
- * pēdējais

laboratorijas darbs. Waspote moduļa pieslēgšana WiFi tīklam.

Darba uzdevumi:

- pirmais
- otrais
- * pēdējais

Apraksts:

Jautājumi atskaitei:

- pirmais
- otrais
- * pēdējais

laboratorijas darbs. Waspote sensoru mērījumu sūtīšana izmantojot GPRS.

Darba uzdevumi:

- * pirmais
- * otrais
- * pēdējais

Apraksts:

Jautājumi atskaitē:

- * pirmais
- * otrais
- * pēdējais

laboratorijas darbs. GPS koordinātu noteikšana un nosūtīšana tīklā.

Darba uzdevumi:

- * pirmais
- * otrais
- * pēdējais

Apraksts:

Jautājumi atskaitei:

- * pirmais
- * otrais
- * pēdējais

laboratorijas darbs. OTAP - over the air programming.

Darba uzdevumi:

- * pirmais
- * otrais
- * pēdējais

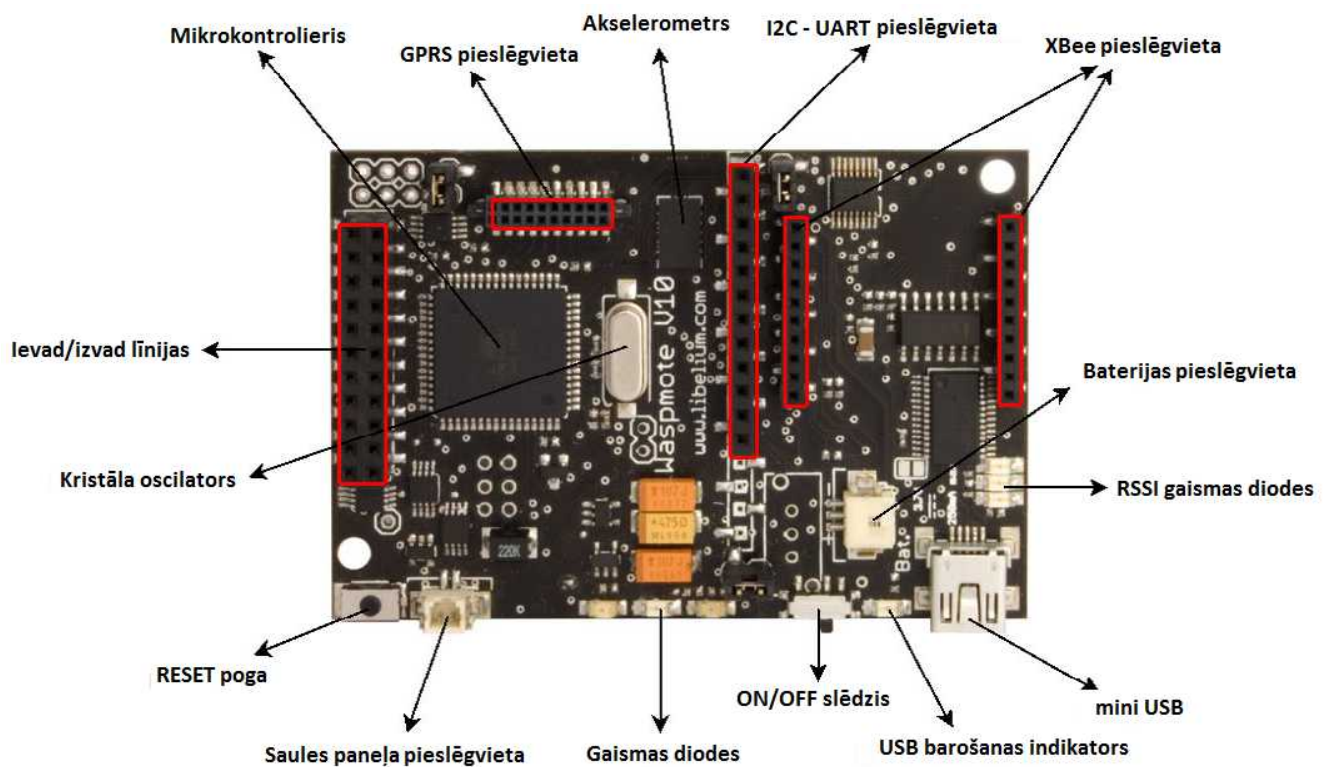
Apraksts:

Jautājumi atskaitei:

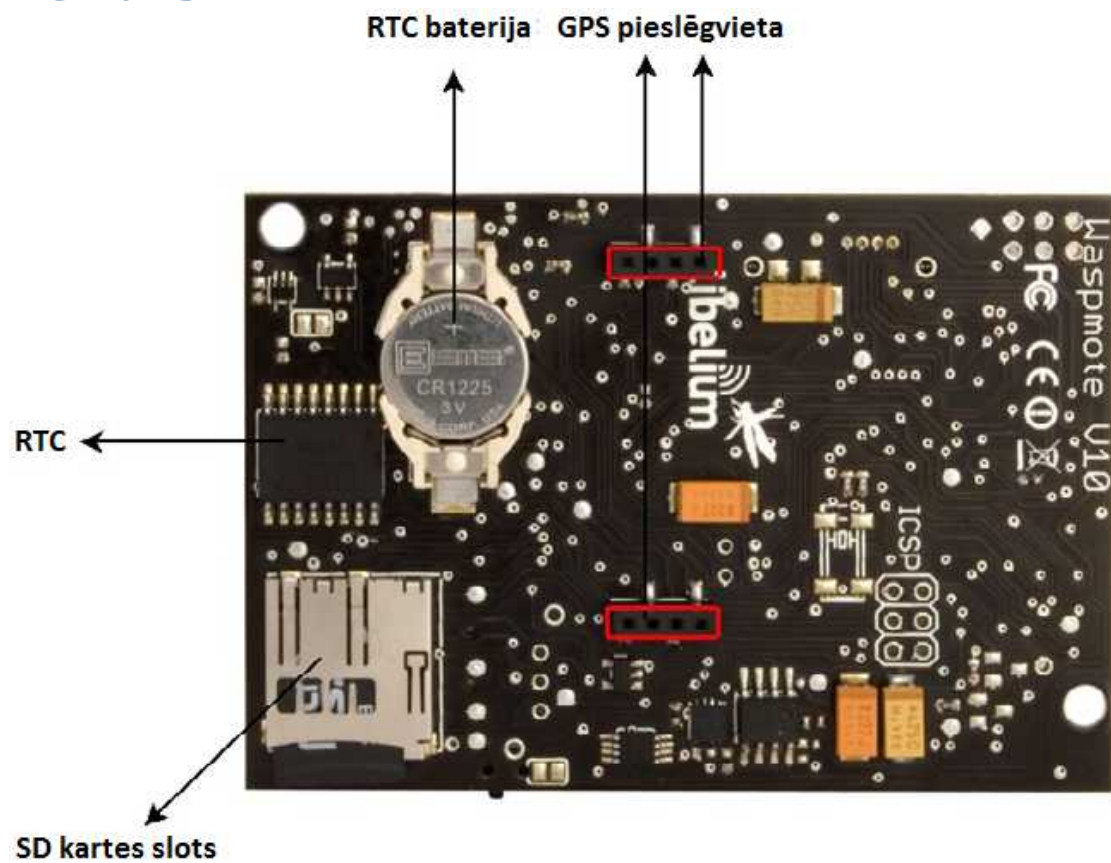
- * pirmais
- * otrais
- * pēdējais

III. Pielikumi

2. Pielikums



Pielikums



Pielikums

DIGITAL8	▪	▪	GND	AUX-SERIAL-1-TX	▪
DIGITAL6	▪	▪	DIGITAL7	AUX-SERIAL-1-RX	▪
DIGITAL4	▪	▪	DIGITAL5	AUX-SERIAL-2-RX	▪
DIGITAL2	▪	▪	DIGITAL3	AUX-SERIAL-2-TX	▪
RESERVED	▪	▪	DIGITAL1	RESERVED	▪
ANALOG6	▪	▪	ANALOG7	GND	▪
ANALOG4	▪	▪	ANALOG5	GND	▪
ANALOG2	▪	▪	ANALOG3	MUX_RX	▪
SENSOR POWER	▪	▪	ANALOG1	MUX_TX	▪
GPS POWER	▪	▪	5V SENSOR POWER	SENSOR POWER	▪
SDA	▪	▪	SCL	SCL	▪
				SDA	▪