Jani Knaappila

Täsmäkieli langattomien moduulien itsenäiseen toimintaan

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

19.4.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Jani Knaappila Täsmäkieli langattomien moduulien itsenäiseen toimintaan 32 sivua + 2 liitettä 19.4.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	
Ohjaaja	lehtori Kimmo Saurén

Insinöörityössä suunniteltiin ohjelmointikieli Bluegigan langattomiin moduuleihin. Langattomat moduulit ovat elektroniikan komponentteja, joita käytetään erilaisissa laitteissa tarjoamaan langaton yhteys. Tälläisiä laitteita ovat esimerkiksi autoradiojärjestelmät, sykemittarit, itkuhälyttimet ja muut laitteet joissa halutaan lyhyen kantaman langatonta tietoliikennettä.

Ohjelmointikielen tavoitteena oli mahdollistaa moduulin itsenäinen toiminta, jolloin ei enää tarvitsisi käyttää ulkopuolista mikro-ohjainta järjestelmän ohjaukseen. Tällöin säästetään lopputuotteen kehitysajassa, kustannuksissa sekä virrankulutuksessa.

Työn alussa haettiin tarvittavat vaatimukset kielelle ja todettiin valmiiden kielten olevan liian raskaita tähän käyttöön tai ne koettiin liian vaikeiksi asiakkaille käyttää. Ratkaisuna suunniteltiin oma kieli vain tähän tiettyyn tarkoitukseen; tällaistä ohjelmointikieltä kutsutaan täsmäkieleksi.

Ohjelmointikieli rakentuu kahdesta osasta: kääntäjästä sekä virtuaalikoneesta. Kääntäjää ajetaan työasemalla, jossa se muuttaa lähdekoodin tavukoodiksi. Tavukoodi ladataan moduulille, jossa virtuaalikone suorittaa kyseistä tavukoodia.

Moduulin ohjaukseen virtuaalikoneesta käytettiin valmista Bluegigan BGAPI-ohjelmointirajapintaa. Tämän seurauksena säästettiin muistivaatimuksissa sekä dokumentaatiossa, koska voitiin hyödyntää aikaisemmin tehtyä työtä.

Työn tuloksena syntynyt täsmäkieli sai nimekseen BGScript. Asiakkaat ovat ottaneet sen hyvin vastaan, ja se onkin käytössä jo yli miljoonassa laitteessa.

Avainsanat	täsmäkieli, sulautettu ohjelmointi, kääntäjä, virtuaalikone, IOT

Author	Jani Knaappila
Title	Domain-specific language for independent operation of wireless modules
Number of Pages	32 pages + 2 appendices
Date	19 April 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	
Instructor	Kimmo Saurén, Senior Lecturer

The goal of this thesis is to describe how a programming language was designed for Bluegiga wireless modules. Wireless modules are electronic subassemblies that provide a wireless interface for devices. These devices can be car audio systems, heartrate monitors or other devices that need short range wireless communication.

The aim of the language is to provide independent operation for modules so that an external controller can be left out. This provides savings in development time, costs and power consumption.

It was found out based on requirements analysis that current programming languages are too heavy in memory usage or too hard to use for customers. The solution was to design a language for this specific task. This kind of language is called domain-specific language or DSL.

This programming language is composed of two parts: a compiler and a virtual machine. The compiler is run on a workstation, and it takes source code as input and provides bytecode as output. The bytecode is then loaded to a module for the virtual machine to run it.

The Bluegiga modules have an application programming interface called BGAPI. The virtual machine was coupled with this interface to control the module from the programming language. This allows reusing of documentation and also lowers module memory requirements.

The programming language that was created as a result of this project is named BGScript. It has been well received by customers and is already in use in over a million devices.

Keywords	programming language, compiler, virtual machine, IOT

Sisällys

Liitteet

Liite 1. BGScript™-käskykanta

Liite 2. Lähdekoodi sykemittariesimerkille

Lyhenteet

ASCII	American Standard Code for Information Interchange, merkistöstandardi
BLE	Bluetooth low energy, matalavirtainen kommunikointistandardi
BGAPI	Bluegiga Application Programming Interface, ohjelmointirajapinta
DSP	Digital Signal Processor, digitaalinen signaaliprosessori
GATT	Generic Attribute Profile, Bluetooth tietokantastandardi
IEEE	Institute of Electrical and Electronics Engineers, tekniikan alan järjestö
SPI	Serial Peripheral Interface, sarjaliikennestandardi
UART	Universal Asynchronous Receiver Transmitter, sarjaliikennestandardi
USB	Universal Serial Bus, sarjaliikennestandardi
XML	eXtensible Markup Language, tiedonkuvausstandardi

1 Johdanto

Bluegigan Technologies Oy:n langattomat moduulit ovat elektroniikan komponentteja, jotka tarjoavat Bluetooth-tai Wi-Fi-yhteyden laitevalmistajien suunnittelemiin tuotteisiin. Moduuleja käyttävä lopputuote on muodostunut kahden tai useamman piirin järjestelmästä, jossa erillinen mikro-ohjain toteuttaa tarvittavan tuotteen toiminnallisuuden ja langaton moduuli tarjoaa langattoman yhteyden ulkomaailmaan.

Jatkuva kilpailu teknologiateollisuudessa luo kustannuspaineita komponenteille. Yksi tapa karsia kustannuksia on tuottaa komponentteja halvemmalla, esimerkiksi siirtämällä tuotantoa edullisempien tuotantokustannusten maihin. Toinen tapa on integroida enemmän toiminnallisuutta yhteen komponenttiin, jolloin voidaan luopua erilliskomponenteista.

Jos yhteen komponenttiin integroidaan paljon toiminnallisuutta, tullaan helposti tilanteeseen, jossa se ei täytä kaikkien asiakkaiden vaatimuksia, jolloin komponentilta vaaditaan räätälöitävyyttä. Tämän räätälöitävyyden hallitseminen sekä integrointiasteen nostaminen on syy työn tekemiseen.

Bluegigan moduulit on toteutettu C-ohjelmointikielellä ja omalla käyttöjärjestelmällä. Jos asiakkaille tarjotaan mahdollisuus kehittää sovelluksiaan suoraan Bluegigan käyttöjärjestelmälle, edellyttää se myös tasokasta teknistä tukea, eikä sellaista pystytä kaikille asiakkaille tarjoamaan.

Työn tarkoituksena on suunnitella menetelmä, jossa laitevalmistajat pystyvät toteuttamaan oman lisäarvoa tuottavan toiminnallisuuden valmiiseen moduuliin. Tällöin erillisestä mikro-ohjaimesta voidaan luopua ja saadaan säästöä kustannuksissa sekä virrankulutuksessa.

2 Bluegigan langattomat moduulit

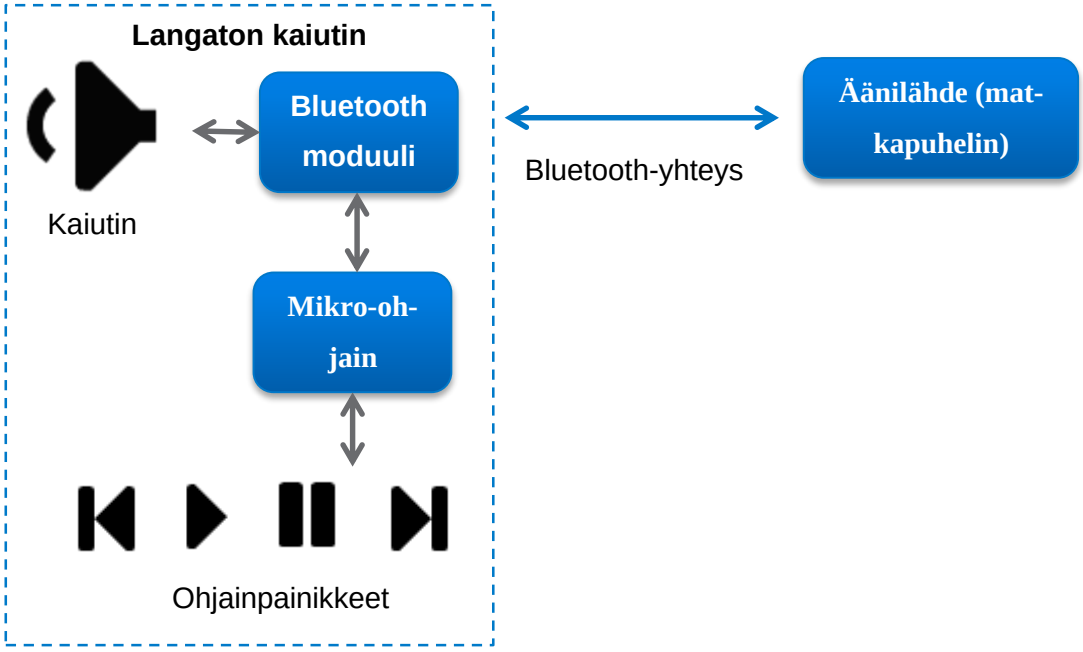
2.1 Langattomien moduulien esittely

Bluegigan langattomat moduulit ovat pieniä älykkäitä langattomien teknologioiden moduuleja, jotka on tarkoitettu vähävirtaisiin järjestelmiin. Langattomat teknologiat kattavat viimeisimmät Wi-Fi ja Bluetooth standardit. Moduulit sisältävät kaiken tarvittavan toiminnallisuuden tarjotakseen langattoman liitynnän isäntälaitteelle. Kuvassa 1 on esitetty kolmen eri teknologian moduuleja.

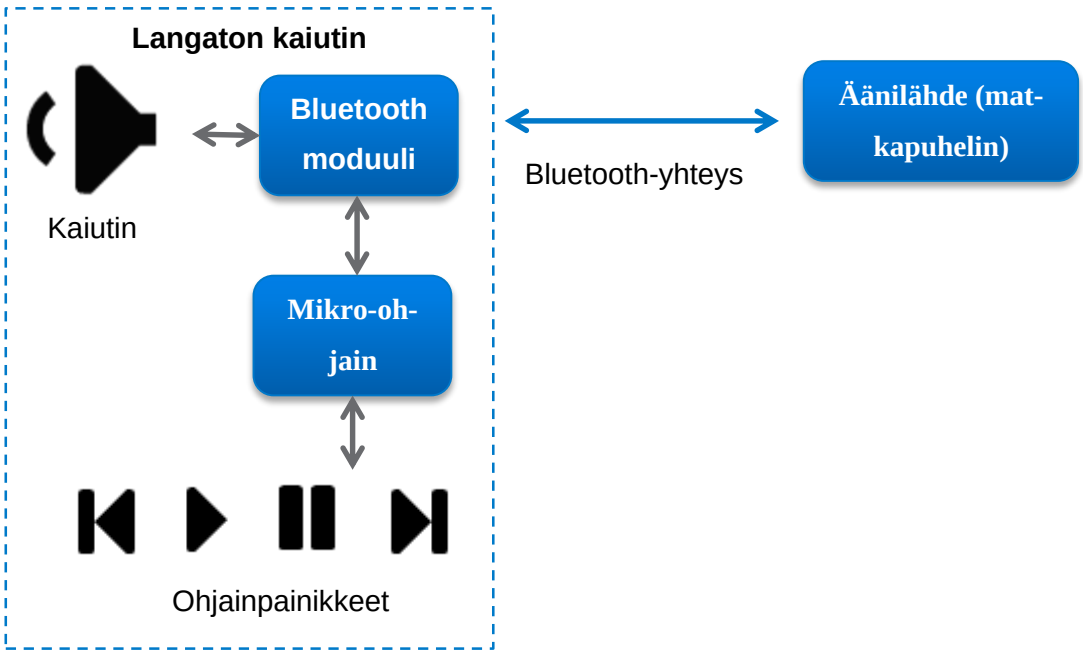
BLE113 Bluetooth 4.0 WF121 Wi-Fi WT32i Bluetooth Audio

Kuva 1. Bluegigan moduuleja (1)

Langattomien moduulien tarkoituksena on helpottaa ja nopeuttaa tuotekehitystä. Esimerkiksi langaton kaiutin voidaan toteuttaa Bluetooth-moduulilla ja ulkoisella mikro-ohjaimella. Tällaisen laitteen lohkokaavio on esitettynä kuvassa

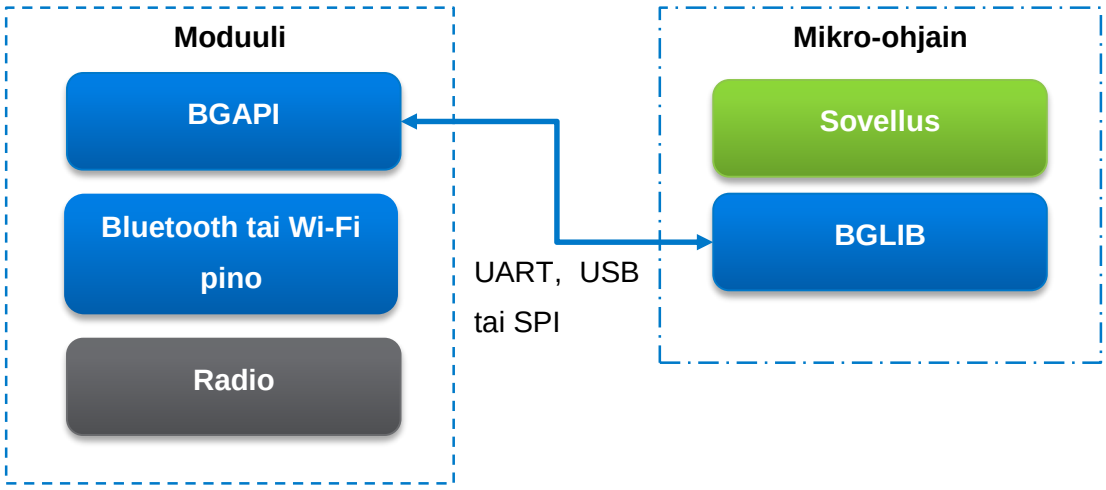


Kuva 2. Bluetooth-moduuli hoitaa Bluetooth-yhteyttä äänilähteen välillä sekä purkaa pakattua audio-dataa DSP:n avulla ääniulostuloksi, joka edelleen ohjataan vahvistimelle ja siitä kaiuttimelle. Mikro-ohjaimen tehtävänä on vain lukea ohjainpainikkeiden tilaa ja välittää ne Bluetooth-moduulille, joka edelleen välittää ne signalointina äänilähteelle. Äänilähde voi tällöin sovelluksesta riippuen esimerkiksi vaihtaa soivaa musiikkikappaletta.



Kuva 2. Langaton kaiutin toteutettuna WT32i Bluetooth moduulilla.

Langattoman moduulin liittyminen isäntäjärjestelmään näkyy kuvassa 3. Moduulin ohjelmisto rakentuu Bluetooth-tai Wi-Fi-pinosta, jota ohjaa BGAPI-ohjelmointirajapinta. BGAPI-rajapintaan liittyy isäntälaite ulkoisen väylän kautta, yleensä mikro-ohjain USB-, SPI- tai UART-väylän kautta. BGLIB on ohjelmointikirjasto C-kielelle, joka tarjoaa valmiita rakenteita BGAPI:n käyttämiseen.

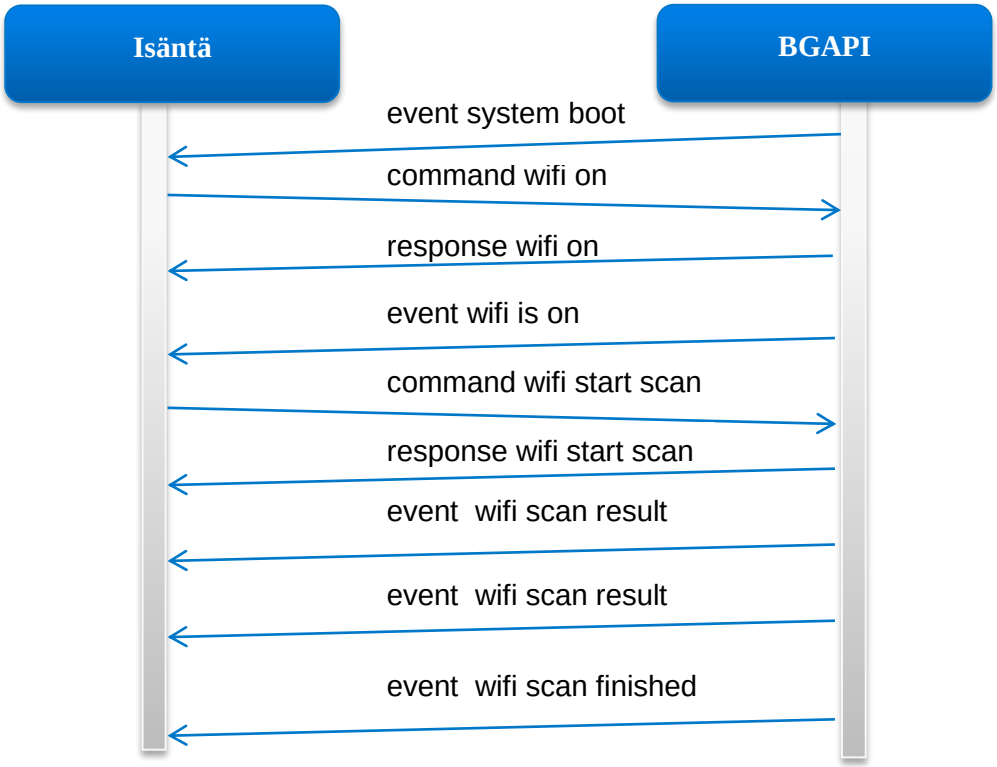


Kuva 3. Bluegigan moduuli ja sen liittyminen isäntäjärjestelmään.

2.2 BGAPI-ohjelmointirajapinta

BGAPI-ohjelmointirajapinta on asynkroninen viestienvälitysprotokolla moduulien ohjaukseen, joka perustuu molemminpuoliseen viestien lähettämiseen.

Isäntä lähettää komennon johon se saa aina vastauksen. Moduuli voi lähettää tapahtumaviestejä tarvittaessa. Kuvassa 4 on esitetty sekvenssikaaviona esimerkki viestien vaihdosta.



Kuva 4. Viestien välitystä BGAPI-rajapinnan avulla WF121 moduulissa. Käynnistyksen jälkeen käynnistetään radio ja sen jälkeen etsitään ympäristöstä tukiasemia.

Jokaiseen komentoon tulee aina vastaus. Tämän tarkoituksena on pitää kaksi erillistä järjestelmää keskenään synkronoituna. Muuten ei voida olla varma, onko viesti mennyt perille ennen seuraavaksi saapuvaa tapahtumaa.

BGAPI-viesti muodostuu otsikosta, joka kertoo tietoa itse viestistä, sekä viestin sisällöstä, joka voi sisältää useampia kenttiä. Useampi tavuisissa kentissä lähetetään vähiten merkitsevä tavu ensin. BGAPI-viestin rakenne esitetty taulukossa 1. BGAPI-viestin sisällössä olevat tietotyypit esitetty taulukossa 2.

Taulukko 1. BGAPI-viestin rakenne.

Kenttä	Bittejä	Kuvaus
Viestin tyyppi	1	0: Command/response 1:Event
Teknologian tyyppi	4	0000: Bluetooth 4.0 single mode 0001: Wi-Fi
Sisällön pituus	11	Sisällön pituus tavuina
Luokan tunniste	8	Message class ID
Viestin tunniste	8	Message ID in class
Sisältö	max 8 * 2048	Sisällön pituus max 2048 tavua

Taulukko 2. BGAPI-viesteissä tuetut tietotyypit.

BGAPI-tietotyyppi	C-tietotyyppi	tavuja
int8	int8_t	1
uint8	uint8_t	1
int16	int16_t	2
uint16	uint16_t	2
int32	int32_t	4
uint32	uint32_t	4
hwaddress	uint8_t[6]	6
uint8array	struct{uint8_t length;uint8_t data[ ];}	≥1

BGAPI-viestit kuvautuvat suoraan C-ohjelmointikielen struktuureihin. Esimerkiksi seuraava heksadesimaalimuodossa esitetty BGAPI-viesti BLE112-moduulilta:

80	0C	00	00	01	00	01	00	01	00	01	00	01	00	01	01

Ensimmäinen tavu kertoo kyseessä olevan Event Bluetooth 4.0 single mode-moduulilta. Ensimmäisen tavun kolme viimeistä bittiä, sekä toinen tavu kertoo viestin sisällön pituuden. Tässä tapauksessa 12-tavua. Viesti on viestiluokasta 0, eli system. Viestin indeksi viestiluokassa on 0, eli boot.

BGAPI-rajapinnan kaikki tietotyypit ovat little endian muodossa, eli esitetään vähiten merkitsevä tavu ensin. Tästä on hyötyä virtuaalikoneessa joka pystyy helposti muuttamaan lukuja eri kokoisten esitysmuotojen välillä lisäämällä tai poistamalla tavuja lukujen lopusta.

BGAPI-dokumentaatiosta (2 s. 186) nähdään, että viesti vastaa listauksessa 1 olevaa C-kielen struktuuria. Kopioimalla viestin sisältö suoraan struktuuriin saadaan struktuurin kentille taulukossa 3 olevat arvot.

```
struct ble_msg_system_boot_evt_t
{
        uint16   major,
        uint16   minor,
        uint16   patch,
        uint16   build,
        uint16   ll_version,
        uint8    protocol_version,
        uint8    hw
}
```

Listaus 1. BLE system boot-eventin C-kielen struktuuri

Taulukko 3. BGAPI-viesti avattuna.

C-tietotyyppi	nimi	arvo
uint16	major_t	1
uint16	minor_t	1
uint16	patch_t	1
uint16	build_t	1
uint16	ll_version_t	1
uint16	protocol_version_t	1
uint8	hw	1

2.3 Asiakaskohtainen räätälöinti

Asiakaskohtaiset räätälöinnit ovat asiakkaan vaatimuksesta tehtyjä muutoksia moduulin perusohjelmistoon. Bluegiga on hoitanut asiakaskohtaiset räätälöinnit ohjelmointityönä maksua vastaan. Tämä on hidasta ja kallista ja tuotekehityksen työpanos on poissa itse tuotteen kehittelystä. Ratkaisuna tähän on, että tarvittavan räätälöinnin hoitaa asiakas itse. Osa räätälöinnistä on erilaisten parametrien säätöä, mikä on helppo hoitaa esimerkiksi konfigurointimahdollisuuksia lisäämällä. Toinen räätälöinnin kohde on toimintalogiikka. Esimerkiksi asiakas voi haluta, että nappia painamalla moduuli avaa automaattisesti yhteyden ulkopuoliseen laitteeseen. Tämänlaiset ongelmat on hoidettu tekemällä ohjelmistoon konfiguroitava käsky, joka suoritetaan, kun I/O muuttaa tilaa. Monimutkaisempi toiminnallisuus on edellyttänyt erillisen mikro-ohjaimen käyttöä.

Ratkaisuna on että asiakas ajaisi omaa ohjelmistoaan itse moduulissa. Yksinkertainen tapa tarjota ohjelmoitavuus asiakkaalle on että Bluegigan ohjelmistopino jaetaan binäärimuotoisena kirjastona, jonka asiakas sisällyttää omaan ohjelmistoonsa mukaan. Tämä edellyttää, että asiakkaalla on kaikki samat kehitystyökalut kuin alkuperäisessä ohjelmistokehityksessäkin. Kääntäjät ja työkalut ovat joissakin tapauksissa hyvin kalliita eivätkä hinnan puolesta sovi kaikille asiakkaille. Myös asiakkaiden osaamisen taso on hyvin vaihtelevaa eikä vaativa C-ohjelmointi kaikilta onnistu.

Asiakkaan kokemia ongelmia ohjelmoitavuudessa voidan vähentää nostamalla abstraktiotasoa ja lisäämällä virtuaalikone käyttöjärjestelmän ja asiakkaan sovelluksen väliin (3 s. 37). Kuvassa 5 esitetyssä järjestelmässä virtuaalikone piilottaa alleen haastavat asiat, kuten muistinhallinnan, virranhallinnan, ohjelmiston päivityksen ja muun vastaavan laitteistotoiminnallisuuden.

Kuva 5. Virtuaalikone sijoittuu sovelluksen ja käyttöjärjestelmän väliin

3 Ohjelmointikielen ohjelmistovaatimukset

3.1 Ei-funktionaaliset vaatimukset

Ohjelmointikielen ohjelmistovaatimukset on kerätty palaverissa, jossa suunniteltiin BGLIB-ohjelmointirajapinta(4). Niistä on tunnistettu tärkeimmät vaatimukset. Lisäksi Bluegigan laitteistoalustat aiheuttavat omia vaatimuksiaan, jotka ovat lueteltu taulukossa 4.

Pienet muistivaatimukset ovat erityisesti BLE-tuotteisiin vaadittu ominaisuus. BLE- tuotteet pohjautuvat 8051-prosessoriin ja siitä on tullut vaatimuksena pieni muistin käyttö. RAM-muistia on käytettävissä ohjelmointikielelle 200 tavua ja Flash-muistia muutama kilotavu.

Helppokäyttöinen kieli sekä myös helppo opittavuus on vaatimus, jotka on haluttu, jotta asiakkaat myös alkaisivat kieltä käyttämään. Käytännössä kielen pitää olla tai pohjautua johonkin yleisesti käyttössä olevaan kieleen.

Nopeasti toteutettavissa oleva ohjelmointikieli haluttiin, jotta saataisiin kieli mukaan BLE-tuotteeseen heti alkuvaiheessa. Nopea kehitysvaatimus aiheuttaa arkkitehtuurivalintoihin sen, että suositaan yksinkertaisia ratkaisuja ja mieluummin karsitaan ominaisuuksissa ja tingitään suorituskyvyssä.

Ylläpidettävyys vaatimuksena haluttiin, koska Bluegigalla on useita erilaisia moduuleja ja tarkoituksena on toteuttaa ohjelmointikieli niihin kaikkiin. Ylläpidettävyys vaatimuksena aiheuttaa sen, että ohjelmointikielen pitää olla sama kaikissa. Ylläpidettävyys aiheuttaa myös sen, että nykyään käytössä oleva BGAPI-ohjelmointirajapinta pitää olla käytettävissä sellaisenaan myös ohjelmointikielessä, jotta ei tarvitsisi toteuttaa uutta rajapintaa.

Taulukko 4. Moduulien käyttämät järjestelmäalustat.

Valmistaja	Microchip	TI	CSR	STMicroelectronics
Piiri	PIC32	CC2540	BlueCore 5	STM32
Ydin	MIPS	8051	XAP2+	Cortex M0
Sanan pituus (b)	32	8	16*	32
FLASH (KiB)	512	128	2048	128
RAM (KiB)	128	8	48	16

*XAP2+ prosessorin pienin tietoyksikkö on 16 bittiä, eli tavun kokokin on 16 bittiä.

3.2 Valmiit ohjelmointikielet

Valmiita ohjelmointikieliä oli tarkoitus hyödyntää ja toteuttaa pelkästään järjestelmäkohtaiset osuudet. Tällöin olisi saatu kaikki etu valmiista ohjelmointikielistä, muun muassa kääntäjä, dokumentoitin, koodiesimerkit ja valmiita kirjastoja eri käyttöihin.

Erilaisia ohjelmointikieliä on lukemattomia eri tarkoituksiin. Seuraavaksi on listattu olennaisimmat sulautettuun käyttöön tarkoitetut ohjelmointikielet sekä niiden hyvät ja huonot puolet tähän tarkoitukseen.

Forth (5) on Chuck Mooren 1970-luvulla kehittämä ohjelmointikieli. Forth vaatii järjestelmältä hyvin vähän. Forth-tulkki perussanastolla ei vie koodimuistia kuin satoja tavuja. Se on myös helppo sovittaa mihin tahansa käyttöön, koska kielen laajentaminen perustuu uusien aliohjelmien lisäämiseen sanastoon. Forth tulkkaa lähdekoodia suoraan konekieleksi, mikä aiheuttaa sen, että kieli on myös hyvin suorituskykyinen.

Forth olisi moduulien ohjelmointikieleksi täydellinen valinta, mutta se ei täytä käytettävyysvaatimusta. Ongelma on, että Forth on pinopohjainen ohjelmointikieli ja sen syntaksi on käänteinen puolalainen notaatio. Tämä nähtiin asiaksi, joka voi olla ihmisille hyvin vaikea asia oppia.

eLua (6) on Lua ohjelmointikielestä kehitetty sulautettavaksi tarkoitettu ohjelmointikieli. eLua on moderni ohjelmointikieli ja käytössä hyvin monessa sovelluksessa. Valitettavasti sen resurssivaatimukset kymmenine kilotavuineen eivät täytä asetettua muistivaatimusta.

TCL (7) on John Ousterhoutin vuonna 1988 julkaisema ohjelmointikieli. TCL on kehitetty sen takia, ettei jokaiseen ohjelmaan tarvitsisi erikseen kehittää omaa ohjelmointikieltä. Vaikka TCL on pieni, se ei aivan pienimpiin sulautettuihin järjestelmiin silti ole sovelias. Pieninkin toteutus vaatii kymmeniä kilotavuja, eikä se siksi täytä muistivaatimusta.

3.3 Valmiit ohjelmointikielet ja vaatimukset

Valmiit ohjelmointikielet eivät täytä teknisiä vaatimuksia. Tämä johtuu pääasiassa siitä, että ne ovat yleiskäyttöisiä ohjelmointikieliä. Jäljelle siis jää tehdä oma ohjelmointikieli, joka on rajoitettu ja suunniteltu tiettyyn tarkoitukseen, jolloin voidaan karsia ominaisuuksia ja sitä kautta muistivaatimuksia.

Pieni muistivaatimus aiheuttaa sen, että ohjelmointikieltä ei voida ajaa tulkattavana moduulissa, vaan se pitää ensin kääntää yksinkertaisempaan muotoon, joka näin vaatii vä-

hemmän resursseja moduulilta. Koska vaatimuksena on myös samanlaisuus eri prosessorialustoilla, on ratkaisuna abstrahoida alustan eroavaisuudet pois. Tietokoneen monitasomallin mukaan voidaan alempi taso abstrahoida virtuaalikoneella. (3 s. 37.)

Vaatimus siitä, että ohjelmointikielen pitää olla nopeasti valmis, on aiheuttanut myös sen, että virtuaalikoneeksi on valittu yksinkertainen pinokone. Virtuaalikone on suunniteltu käyttämään olemassa olevaa BGAPI-rajapintaa, jotta ylläpidettävänä olisi vain yksi rajapinta.

Koska vaatimuksena on helppokäyttöisyys, päätettiin ohjelmointikielen syntaksi perustaa BASIC-ohjelmointikieleen. Kyseinen ohjelmointikieli on suunniteltu helppokäyttöiseksi (8). BASIC-kielen syntaksi on hyvin suosittu myös monessa muussa kielessä, joten oletettavasti sen pitäisi olla tutumpi ympäristö käyttäjille.

Ohjelmointikieleksi on valikoitunut BASIC-syntaksiin perustuva kieli, joka käännetään PC:llä ensin tavukoodiksi ja jota ajetaan moduulissa pinopohjaisessa virtuaalikoneesssa käyttäen BGAPI-rajapintaa.

4 Syntaksi

Ohjelmointikielen syntaksi perustuu BASIC-ohjelmointikieleen, eli jokainen rivi on jokin kielen komento. Syntaksin esimerkki näkyy listauksessa 2.

Koska kielen pitää olla yhteensopiva BGAPI:n kanssa, sen pitää myös tukea samoja tietotyyppejä kuin BGAPI. BGAPI:n tietotyypit on esitetty taulukossa 2. Pyrkimyksenä on pitää ohjelmointikieli mahdollisimman yksinkertaisena, joten muuttujana käytetään pelkästään 32-bittistä kokonaislukua. Tällä pystytään esittämään suurin osa tietotyypeistä. Tietotyyppejä "hwaddress" sekä "uint8array" varten tarvitaan tuki myös tavualkioista taulukkoa varten. Muuttujat ja taulukot esitellään kielessä DIM-sanalla.

Koska BGAPI perustuu moduulin lähettämiin tapahtumiin, voidaan ohjelmointikielessä tehdä kaiken suorituksen olemaan vastauksena näihin tapahtumiin. Kielen EVENT-sanalla esitellään rutiini, joka suoritetaan, kun moduuli lähettää tapahtuman.

Moduulin ohjaamiseen tarvittavia BGAPI-komentoja voidaan lähettää ohjelmointikielestä CALL-sanalla. BGAPI-vastaukset tulevat tähän CALL-sanaan vastauksena.

Jotta kieleen saadaan ehdollista toimintaa, tarvitaan IF-sana sekä silmukoita varten WHILE-sana.

#-merkkiä käytetään rivin alussa merkitsemään kommenttia.

```
event system_boot(major, minor, patch, build, ll_version, protocol, hw)
    # Initialize the display(see NHD - C0216CZ - FSW - FBW - 3V3 data sheet)
    call hardware_io_port_write(1, $7, $1)
    call hardware_io_port_config_direction(1, $7)
    call hardware_spi_transfer(0, 11,
"\x30\x30\x30\x39\x14\x56\x6d\x70\x0c\x06\x01")
    call hardware_io_port_write(1, $7, $3)
    # Write "Hello, world" to the display.
    call hardware_spi_transfer(0, 12, "hello, world")
end
```

Listaus 2. Esimerkkiohjelma joka BLE112-moduulin käynnistyessä alustaa SPI-väylään kytketyn näytön ja kirjoittaa "hello, world" näytölle.

5 Kääntäjä

5.1 Kääntäjän tehtävä

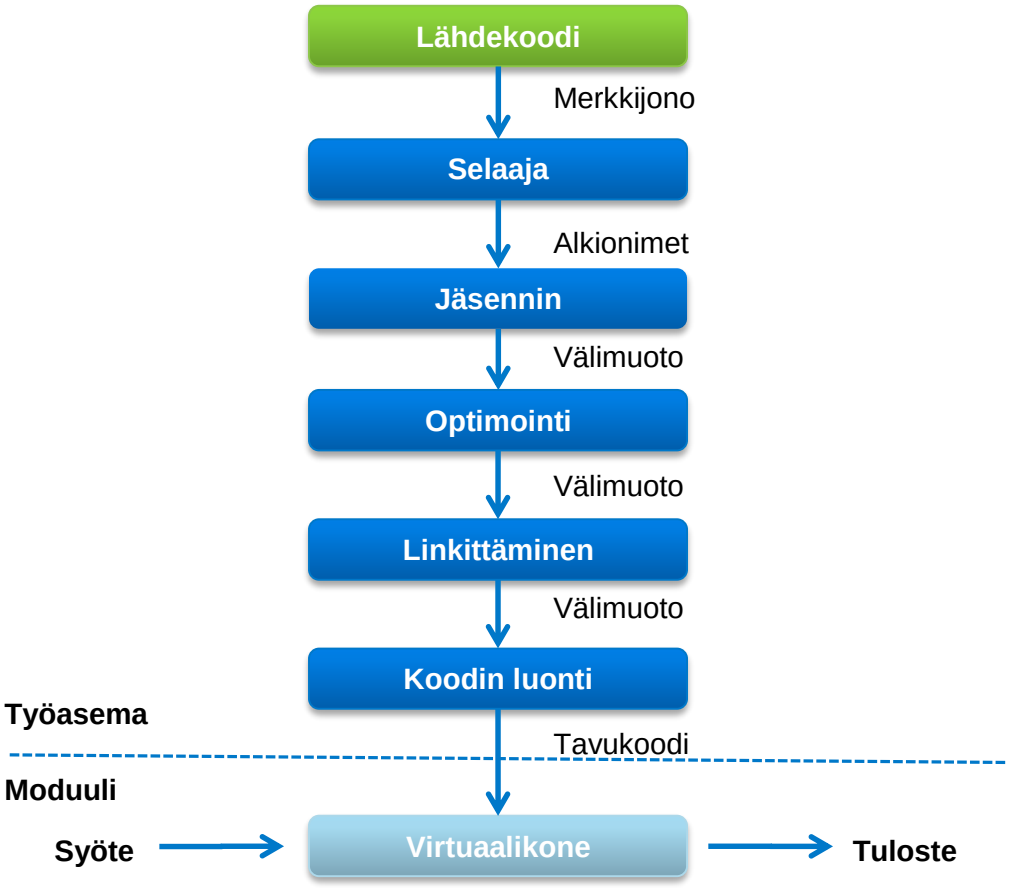
Kääntäjän tehtävänä on muuttaa lähdekoodia muotoon, joka on tietokoneella ajettavissa. Kääntäjä on jaettu erillisiin kokonaisuuksiin, joista kukin edelleen muokkaa koodia seuraavalle kokonaisuudelle.

BGScript-kääntäjä on hybridikääntäjä (9 s. 3.) joka kääntää lähdekoodia ensin tavukoodiksi ja joka edelleen ajetaan virtuaalikoneessa. Kääntäjän arkkitehtuuri on esitetty kuvassa 6.

Työasemassa kirjoitetaan tekstieditorilla lähdekoodi ja se annetaan syötteenä kääntäjäohjelmalle. Kääntäjä ensimmäisessä vaiheessa muuttaa lähdekoodin alkionimiksi. Alkionimet annetaan eteenpäin jäsentimelle, jossa kielen rakenteen mukaan tulkitaan alkionimiä ja niistä muodostetaan välimuodon esitys. Välimuodossa on jo olemassa tieto, minkälaista koodia siitä tullaan luomaan.

Välimuoto annetaan optimointivaiheelle, jonka tehtävänä on muuttaa koodia siten, että se vie mahdollisimman vähän tilaa. Tämän jälkeen se annetaan linkitysvaiheelle, joka asettaa välimuodon esityksen viitteille oikeat osoitteet. Lopuksi tästä välimuodon esityksestä luodaan varsinainen tavukoodi.

Saatu tavukoodi syötetään moduuliin. Moduulissa ajetaan virtuaalikonetta, joka suorittaa tavukoodia. Virtuaalikone saa oman syötteensä BGAPI-tapahtumista ja antaa tulosteena BGAPI-käskyjä.



Kuva 6. Hybridikääntäjä.

5.2 Selaaja

Selaajan tehtävänä on lukea lähdekooditiedostosta tekstiä rivi kerrallaan ja muuttaa yksittäisiä sanoja alkionimiksi. Alkionimessä on tunniste, minkä tyyppinen alkionimi se on, sekä määrite lisätiedosta kyseiselle alkionimelle. Esimerkiksi luku-alkionimellä on tieto, mikä luku on varsinaisesti syötetty. Samoin tekstijonoilla on tieto, mikä tekstijono on syötetty. Lisäksi alkionimissä on myös tieto, mistä lähdekooditiedostosta ja miltä riviltä se on peräisin, jotta virheilmoitukset olisivat havainnollisempia.

Tekstisana

Tekstisana on mikä tahansa sana, joka muodostuu tekstistä. Näitä ovat avainsanat, kuten WHILE ja EVENT, sekä muuttujien ja komentojen nimet.

Tekstisanat tunnistetaan ASCII-merkkien perusteella. Jos merkki on jokin aakkosiin kuuluva merkki, aloitetaan lukemaan sanaa sisään ja lopetetaan, jos merkki ei enää kuulu aakkosiin. Jos näin saatu sana on jokin esimääritelty sana, lisätään alkionimeksi sitä vastaava alkionimi. Muuten luodaan yleisteksti-alkionimi. Jäsennin käsittelee yleisteksti-alkionimeä seuraavassa vaiheessa riippuen asiayhteydestä esimerkiksi funktiona tai muuttujana.

Luku

Luku-alkionimi on mikä tahansa desimaali- tai heksadesimaali-luku. BGScript hyväksyy lukuja desimaalimuodossa, ja käyttäen $-merkkiä voidaan syöttää myös heksadesimaalilukuja. Esimerkiksi $55 on heksadesimaaliluku, jota vastaa desimaalilukuna luku 85.

Luku tunnistetaan, jos luku alkaa merkillä, joka vastaa numeroita 0- 9 tai $-merkillä. Lukua luetaan sisään merkki kerrallaan, kunnes tulee vastaan merkki, joka ei ole numero. Näistä luodaan numeroalkionimi, jonka arvoksi laitetaan saatu luku.

Tekstijono

Tekstijonot ovat tapa syöttää merkkijonoja ohjelmaan ilman, että niitä pitää ensin alustaa taulukkoon.

Tekstijonot tunnistetaan lainausmerkeistä. Kun selaaja lukee lainausmerkin, se alkaa lukemaan tekstiä sisään, kunnes tulee seuraava lainausmerkki. Selaaja tukee myös C-kielen tapaa syöttää erikoismerkkejä kenoviivan avulla. Tämän jälkeen se luo tekstivakio-alkionimen ja antaa sen arvoksi saadun tekstijonon.

Erikoismerkit

Erikoismerkit, esimerkiksi erilaiset matemaattiset operaattorit, muuttuvat suoraan omiksi alkionimikseen.

Unaariset operaattorit

Unaarinen operaattori on operaattori, joka ottaa vain yhden parametrin. BGScript ei tällä hetkellä tue näitä.

Negatiiviset luvut tulkitaan selausvaiheessa. Selaajalla on logiikka, jonka mukaan se tulkitsee, tarkoittaako negatiivinen merkki vähennyslaskua vai negatiivista lukua. Tämä toimii lukuvakioilla hyvin, mutta muuttujilla se ei enää toimi. Esimerkiksi seuraavanlaista käskyä ei tueta A= - A. Tämä voidaan toki kiertää antamalla A=0 - A

Oikeampi tapa olisi ollut tehdä tuki unaarisille operaattoreille, jolloin negatiivinen luku olisi ollut itse asiassa kaksi alkionimeä: Unaarinen operaattori, joka antaa seuraavan luvun vastaluvun, ja itse luku.

Unaarista operaattoria tarvittaisiin myös boolean ja binäärisessä logiikassa. Esimerkiksi C-kielen operaattorit ! ja ~.

5.3 Jäsennin

Selaajan luomat alkionimet syötetään jäsentimelle. Jäsentimen tarkoituksena on muuttaa alkionimet väliesitysmuodoksi, josta voidaan muodostaa varsinainen lähdekoodi.

Jäsennin on tehty tilakoneeksi. Jäsentimellä on kaksi tilaa: perustila, jossa määritellään muuttujia ja tapahtumia, sekä tapahtumatila, jossa määritellään itse ohjelmakoodia.

Perustila

Perustila on jäsentimen tila, jossa määritellään muuttujia ja taulukoita DIM-alkionimellä ja tapahtumia EVENT-alkionimellä.

Esimerkiksi EVENT-alkionimen käsittely on seuraavanlainen prosessi:

1. Luetaan seuraava alkionimi ja tarkistetaan, onko se vapaa tekstisana.

2. Tarkistetaan onko tapahtuma määritelty XML-tiedostossa ja luodaan sille tapahtumankäsittelijä ulos menevään jonoon.

3. Luetaan seuraava alkionimi ja odotetaan sen olevan vasen kaarisulje merkki.

4. Luetaan tekstisana sisään, tulkitaan se parametriksi ja luodaan sille merkintä symbolitauluun. Parametri saa tyyppinsä XML-tiedostosta.

5. Luetaan seuraava alkionimi, ja jos kyseessä on pilkku, siirrytään kohtaan 4. Jos taas kyseesssä on oikea kaarisulje merkki, varmistetaan, että syötetyllä tapahtumalla on yhtä monta parametriä kuin XML-tiedostossa on määritelty.

6. Siirrytään tapahtumatilaan.

Tapahtumatila

Tapahtumatila on jäsentimen tila, jossa luodaan varsinaista ohjelmakoodia.

Esimerkiksi jos seuraava alkionimi on tekstisana ja se on aiemmin määritelty DIM-sanalla muuttujaksi, jäsennin olettaa, että kysymyksessä on sijoitus muotoa a=x+y. Tällöin se laittaa ulos menevään jonoon muuttujan tunnisteen ja odottaa seuraavaksi alkioksi ”yhtä suuri kuin”-merkkiä (=). Sen saatuaan se olettaa, että seuraavaksi pitää saada lauseke, jonka tuloksena on luku. Lopuksi se luo ulos menevään jonoon operaation ”WRITE”. (WRITE-operaatio virtuaalikoneessa ottaa pinosta aiemmin luodun luvun ja kirjoittaa sen pinossa olevaan muuttujan osoitteeseen.)

END-alkionimellä siirrytään takaisin perustilaan.

Lauseke

Lauseke on yhdistelmä lukuja ja operaattoreita. Virtuaalikoneen suoritettua lausekkeen muodostaman tavukoodin saa se tuloksena pinoon yhden luvun. Lauseketta voidaan täten käyttää jokaisessa tilanteessa, jossa voitaisiin hyväksyä luku. Yksinkertaisimmillaan lauseke voi olla esimerkiksi yksi vakio.

Koska lausekkeet syötetään infix-notaatiolla, ne joudutaan muuttamaan postfix-notaatioon. Tähän käytetään shunting-yard-algoritmia (10 s. 7.). Matemaattisten operaattoreiden laskentajärjestys perustuu C-ohjelmointikielen järjestykseen. Lausekkeen käsittelijä lukee alkionimiä jonosta ja käyttäen shunting-yard-algoritmia jäsentelee ne uudelleen, antaen tuloksena väliesitysmuodon, joka tuottaa yhden luvun pinoon.

Esimerkiksi lauseke infix-notaatiolla on muotoa 1+2*a, tulee siitä postfix-notaatiolla muotoa: 2,a,*,1,+.

5.4 Väliesitysmuoto

Väliesitysmuotoja voi olla kahta muotoa: abstrakti sekä konkreettinen esitysmuoto. Abstrakti esitysmuoto esitetään yleensä abstraktina syntaksipuuna (9 s. 41.). Abstraktissa muodossa ei ole mitään ylimääräistä tietoa, miten koodi pitäisi esittää. Tämä helpottaisi muun muassa optimointia ja kielen laajentamista tulevaisuudessa, mutta abstraktin syntaksipuun toteuttaminen vaatisi enemmän työtä. Yksinkertaisemman toteutuksen takia tässä työssä ei siis käytetä abstraktia esitysmuotoa vaan konkreettista esitysmuotoa. Tässä tapauksessa se tarkoittaa, että väliesitysmuoto sisältää jo tietoa siitä miten koodi pitää luoda lopulliseksi tavukoodiksi. Lisäksi tässä työssä konkreettisen väliesitysmuodon tietorakenne on lista, sitä voisi siis kutsua konkreettiseksi syntaksilistaksi.

5.5 Optimointi

Optimoinnin tarkoituksena on muokata ohjelmakoodia tehokkaammaksi. Tehokkuus voi olla joko tavukoodin koon pienentämistä, jolloin se vie vähemmän muistia, tai sitten käyttää tehokkaampia käskyjä, jolloin koodi suorittuu nopeammin. BGScriptissä suorituskyky ei ole vaatimuksissa, jolloin keskitytään tavukoodin koon pienentämiseen.

Käytännössä tämä tapahtuu tunnistamalla tapauksia, joissa voidaan poistaa koodia. Esimerkiksi komento A=1+2 voidaan muuttaa muotoon A=3.

Optimoija käy läpi ohjelman väliesitysmuotoa ja tunnistaa tiettyjä ennalta määriteltyjä rakenteita, joita voidaan muuttaa lyhyemmiksi muodoiksi. Optimointitapaa kutsutaan nimellä peephole optimization (9 s. 549.).

5.6 Linkittäminen

Linkittämisen tarkoituksena on yhdistää käskyjä toisiinsa, jos niiden toiminta on riippuvaisia jonkin toisen käskyn sijainnista. Esimerkiksi hyppykäskyn ja ehtokäskyn pitää tietää, mihin kohtaan ohjelmaa pitää hypätä. Tämä täytyy tehdä viimeiseksi, koska ohjelman koko muuttuu optimoidessa.

```
if input=2
    input = 3
end if
```

Listaus 3. Esimerkki IF-käskystä.

Listauksessa 3 IF-käskyn tarvitsee tietää, mihin osoitteeseen pitää hypätä jos ehto on epätosi. Tätä ei ole voitu tehdä jäsennysvaiheessa, koska ei tiedetä minkä kokoinen tosiehdon koodipolku on. Siksi tämä tehdään linkitysvaiheessa käymällä väliesitysmuotoa läpi ja etsimällä IF-käskyt ja asettamalla hyppyosoitteeksi niitä vastaavat ENDIF- tai ELSE-käskyt.

Linkitysvaiheessa luodaan myös hyppytaulukot kaikille tapahtumakäsittelijöille. Hyppytaulukot luodaan kaksiosaisiksi XML-tiedoston perusteella, ensin taulukot viestien luokille ja seuraavaksi luokittain viestien indekseille.

5.7 Koodin luonti

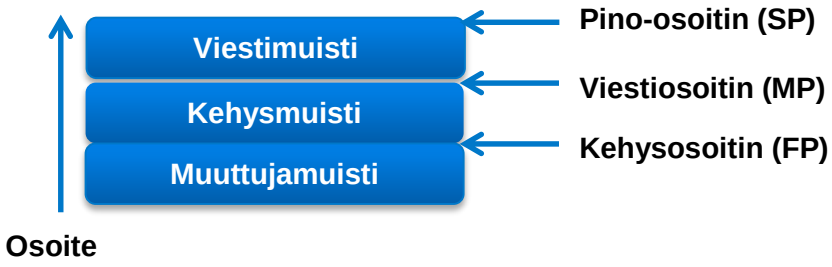
Lopuksi kääntäjä käy läpi koodin väliesitysmuodon ja muuttaa sen suoraan tavukoodiksi. Tuloksena syntyy binäärimuotoinen tavukoodi, joka voidaan ladata virtuaalikoneeseen.

6 Virtuaalikone

Virtuaalikone on toteutettu moduulin ohjelmistoon. Se lukee tavukoodia muistista ja suorittaa koodin mukaisia käskyjä. BGScriptissä emuloidaan ohjelmallisesti tietokonetta, jolla on seuraavat ominaisuudet:

- Lukee tavukoodia ja pystyy vaihtamaan suoritusosoitetta tavukoodissa.
- Sisältää pinon, jonka päällimmäisiä lukuja voidaan operoida ja laittaa vastaus takaisin pinoon.
- Kehysosoitin pinoon, joka osoittaa, mistä alkaa tämänhetkinen alirutiinin pino.
- Viestiosoitin pinoon, joka osoittaa, mistä alkaa BGAPI-viesti jota ollaan rakentamassa.
- Muistia muuttujille ja taulukoille.

Virtuaalikoneen muisti ja osoittimet ovat esitetty kuvassa 7.



Kuva 7. Virtuaalikoneen muistialueet ja osoittimet. Pino kasvaa tässä kuvassa ylöspäin.

6.1 Tietotyypit

Virtuaalikoneessa on kaksi tietotyyppiä: 32-bittinen etumerkillinen kokonaisluku, joka on tallennettu little endian-muodossa, sekä 8-bittinen etumerkitön luku, josta muodostetaan taulukoita. Tietorakenteita on vain yksi, taulukko joka muodostuu edellämainituista 8-bittisistä alkioista.

Virtuaalikone kykenee käsittelemään taulukkoa myös kokonaislukuna. Jos annettu taulukko on alle 4 tavua, se laajentaa sen automaattisesti kokonaisluvuksi lisäämällä 0-tavuja pinoon. Samoin pinosta pystytään siirtämään lukuja alle nelialkioiseen taulukkoon, jolloin virtuaalikone jättää ylimääräiset tavut huomioimatta.

Erilaiset vakiot ovat upotettuna suoraan tavukoodiin. Ne seuraavat vakiota käyttävää tavukoodi-alkiota.

6.2 Tavukoodin suoritus

Virtuaalikone suorittaa tavukoodia vastineena BGAPI-tapahtumaan. Ohjelmalaskuri alustetaan aina saadun tapahtuman mukaan oikean käsittelijän alkuun.

Virtuaalikone suorittaa tavukoodia lukemalla muistista ohjelmalaskurin osoittamasta paikasta tavukoodin ja suorittamalla tavukoodia vastaavan käskyn. Luettelo tavukoodeista on työn liitteenä 1. Kaikki tavukoodin käskyt operoivat käyttäen pinoa, muuttujamuistia tai kehysmuistia.

Pinon suora operointi

Pinon suoraan operointiin on käskyjä, joilla laitetaan vakioita pinoon, poistetaan pinosta tavuja tai vaihdetaan päällimmäisiä lukuja pinossa.

Suurin osa tavukoodista on vakioiden laittamista pinoon, nämäkin ovat enimmäkseen lukuja jotka mahtuvat yhteen tavuun. Tätä on optimoitu tekemällä käsky (32BIT8), joka lukee yhden tavun ja laittaa sen neljän tavun lukuna pinoon.

Tavujen poistamista pinossa käytetään muokkaamaan lukuja sopimaan BGAPI-tietotyyppeihin.

Päällimmäisten lukujen vaihtamista käytetään helpottamaan kääntäjän suunnittelua. Joissakin tilanteissa joudutaan laittamaan lukuja eri järjestyksessä pinoon. Tällöin voidaan laskea luvut ”väärässä” järjestyksessä pinoon ja laittaa lopuksi SWAP-käsky, jolla ne vaihdetaan oikein päin.

Operaattorit

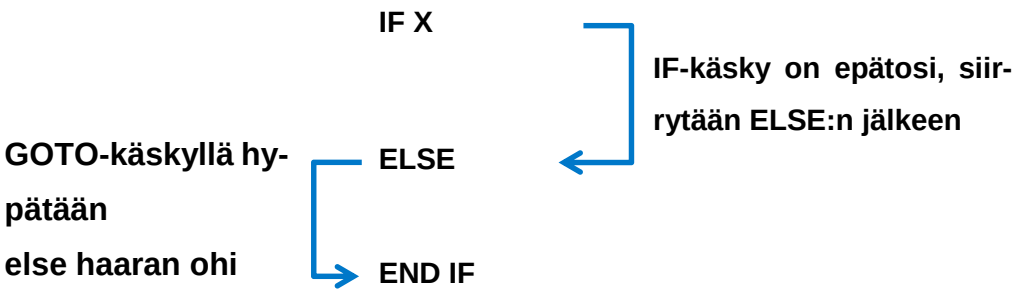
Operaattorit voivat olla matemaattisia operaattoreita, binäärisiä operaattoreita tai erilaisia ehtolausekkeita. Yhteistä kaikille on se, että ne ottavat kaksi lukua pinosta ja laittavat operaattorin tuloksen pinoon.

IF-käsky

IF-käsky ottaa pinosta yhden luvun, ja jos luku on epätosi, se lukee tavukoodista seuraavat kaksi tavua ja asettaa ohjelmalaskurin niiden muodostamaan osoitteeseen. Tällä käskyllä tehdään ehdollisia rakenteita ohjelmaan.

GOTO-käsky

GOTO-käsky ottaa pinosta kaksi lukua ja asettaa ohjelmalaskurin näiden muodostamaan osoitteeseen. GOTO-käskyä käytetään IF-ELSE-rakenteen ELSE-haaran muodostamiseen. Tapahtumaa havainnollistettu kuvassa 8.



Kuva 8. IF-ELSE rakenne.

Muistien käsittely

Muistien käsittelyyn on omat käskynsä, koska BGScriptissä on kolmea erilaista muistia; pino-, muuttuja- ja kehys-muistia. Tavukoodissa on käskyjä, joilla tietoa voidaan siirtää niiden välillä.

Koska pino on ainoa muisti, jota operaattorit voivat käyttää, joudutaan muuttujamuistista aina siirtämään tietoa pinoon, jotta sitä voidaan käyttää. Tähän on käskyjä, jotka siirtävät joko lukuja tai taulukoita pinoon. Vastaavasti pinosta voidaan siirtää lukuja ja taulukoita muuttuja muistiin.

Kehysmuisti on muisti, jossa on tieto, joka liittyy BGAPI-tapahtumiin tai vastauksiin. Kehysmuistista voidaan siirtää lukuja ja muuttujia pinoon.

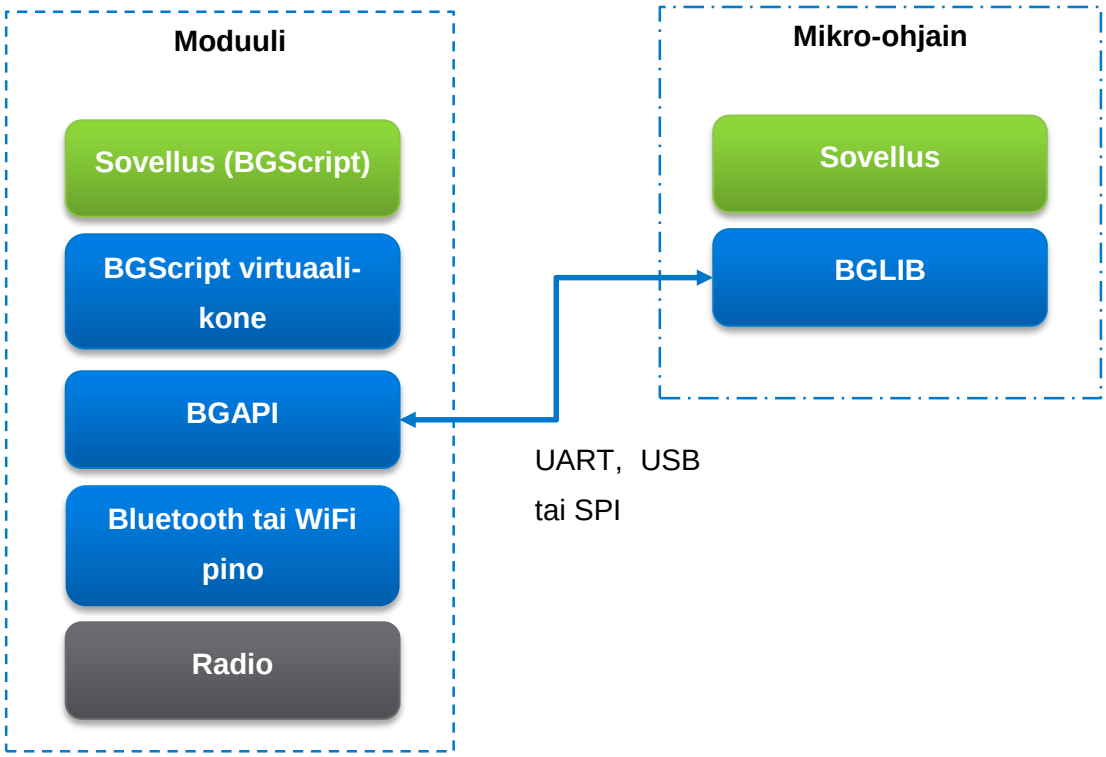
Erikoiskäskyt

Erikoiskäskyt on tarkoitettu tiettyyn tehtävään. Näiden tarkoituksena on tarjota optimoituja käskyjä usein tehtäviin tapahtumiin. Näitä ovat esimerkiksi muistin kopiointi taulukoiden välillä MEMCPY-käskyllä, taulukoiden alustus MEMSET-käskyllä sekä IEEE FLOAT muodon esittäminen, jota tarvitaan tietyissä lääketieteellisissä sovelluksissa.

6.3 Virtuaalikoneen BGAPI-yhteys

Virtuaalikoneen BGAPI-yhteys eli virtuaalikoneen liittyminen Bluegigan ohjelmistopinoon kuvattuna kuvassa 9.

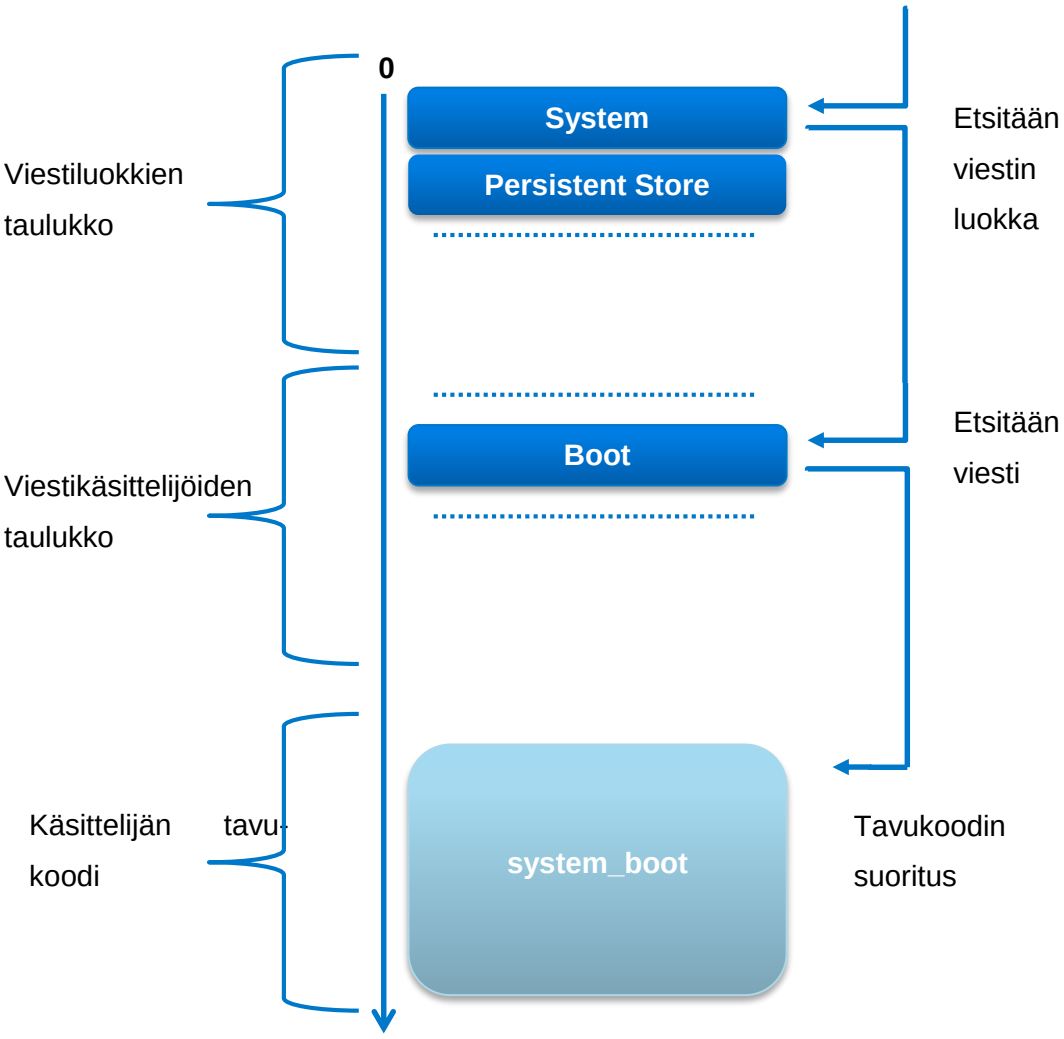
BGAPI-tapahtumat lähetetään BGAPI:lta virtuaalikoneelle. Vastauksena näihin tapahtumiin virtuaalikone lähettää BGAPI-komentoja takaisin BGAPI:lle, joilla ohjataan moduulin toimintaa. Näihin käskyihin liittyvät vastaukset lähetetään BGAPI:lta takaisin virtuaalikoneelle.



Kuva 9. BGScript Bluegigan ohjelmistopinossa.

Moduulin ohjelmistopino lähettää BGAPI-viestejä myös BGScript-virtuaalikoneen viestinkäsittelijälle. Virtuaalikoneen viestinkäsittelijä katsoo viestistä viestin tyypin, ja jos kyseessä on tapahtuma, niin etsii oikean käsittelijän viestin luokan ja indeksin perusteella. Viestin luokan perusteella etsitään oikea viestikäsittelijöiden taulukko. Tästä viestikäsittelijätaulukosta etsitään viestin tunnisteen perusteella oikea käsittelijä, jonka tavukoodia aletaan suorittamaan. Tapahtumaa on havainnollistettu kuvassa 10.

Jos käsittelijä on määritelty, ohjelmalaskuri alustetaan osoittamaan tavukoodin alkuun, kehysosoitin alustetaan osoittamaan pinon tämänhetkiseen kohtaan ja viesti kopioidaan virtuaalikoneen pinoon. Kehysosoitin osoittaa tällöin käytännössä viestin sisältöön. Vastaavasti pino-osoitin laitetaan osoittamaan viestin jälkeiseen osoitteeseen. Virtuaalikone pystyy tällöin kehysosoittimen kautta pääsemään käsiksi saatuun viestiin.



Kuva 10. BGScript viestin käsittelijän etsiminen.

Esimerkiksi listauksen 2 tapahtuman käsittelijässä "system_boot", pinon sisältö olisi taulukon 5 mukainen.

Taulukko 5. Pinon sisältö system_boot käsittelijässä suhteessa kehysosoittimeen.

Osoite	Tietotyyppi	Nimi
0–1	uint16	major
2–3	uint16	minor
4–5	uint16	patch
6–7	uint16	build

8–9	uint16	ll_version
10	uint8	protocol
11	uint8	hw

Jotta virtuaalikone kykenisi ohjaamaan moduulin toimintaa, se lähettää BGAPI-komentoja BGAPI:lle. Kun kielessä on käytetty CALL-sanaa, esimerkiksi listauksen 2 rivillä 3. Virtuaalikone aloittaa viestin rakentamisen omaan pinoonsa työntämällä pinoon viestiosoittimen sekä kehysosoittimen ja asettaa viestiosoittimen osoittamaan pinon tämänhetkiseen osoitteeseen. Tämän jälkeen se suorittaa käskyjä, jotka rakentavat viestin sisällön pinoon. Nämä voivat olla monimutkaisiakin matemaattisia lausekkeita.

Kääntäjä on kääntäessään lähdekoodia tunnistanut viestin ja sen sisältämät parametrit. Koska tietotyypit ovat kielessä sisäisesti 32-bittisiä, kääntäjä on lisännyt käskyjä, jotka tiputtavat ylimääräiset tavut pois pinosta jos tietotyyppi on pienempi kuin 32-bittiä. Esimerkin tapauksessa pino olisi seuraavanlainen alkaen viestiosoittimesta:

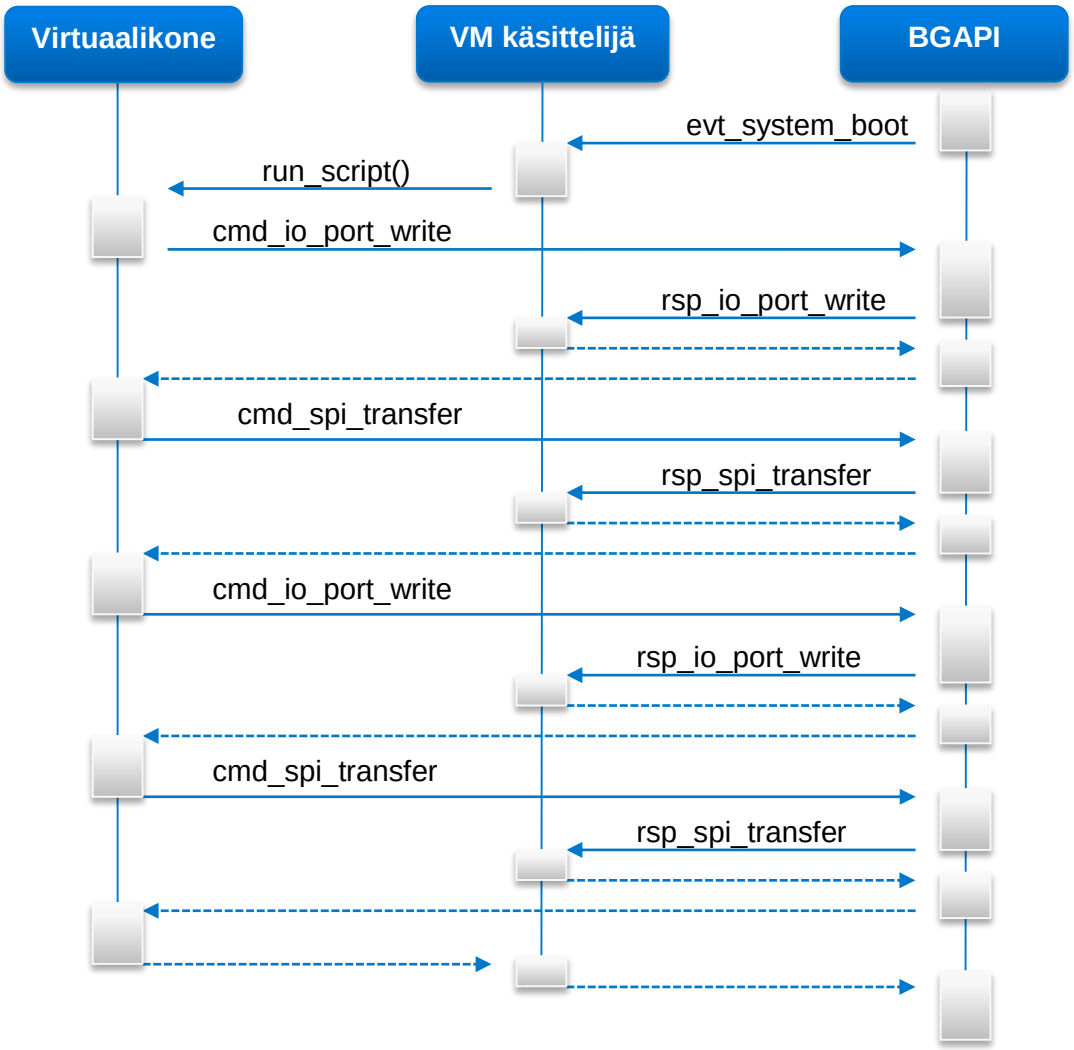
osoite	data
0	1
1	7
2	1

Kun viesti on rakennettu, suoritetaan tavukoodi, joka lukee seuraavasta tavusta käskyn indeksin, kutsuu BGAPI -viestinkäsittelijää antaen sille osoitteeksi viestiosoitteen sekä käskyn indeksin.

Koska virtuaalikone sekä BGAPI-viestinkäsittelijä toimivat samassa prosessissa, niin suoritus ei palaudu virtuaalikoneelle ennen kuin viesti on käsitelty ja sille on lähetetty vastaus.

Vastaus toimii samalla lailla kuin tapahtuman käsittelykin, eli BGAPI-viestinkäsitteljä lähettää viestin BGScript viestinkäsittelijälle. Viestinkäsittelijä asettaa kehysosoittimen pinon tämänhetkiseen osoitteeseen ja laittaa vastauksen pinoon, mutta palaakin tämän jälkeen takaisin lähettäjäfunktioon. Seuraavaksi BGAPI-viestinkäsittelijä palaa itse myös

takaisin kutsujalle, jolloin palataan takaisin virtuaalikoneeseen. Listauksen 2 funktiokutsut on esitetty kuvassa 11.



Kuva 11. Virtuaalikoneen ja BGAPI-käsittelijän väliset funktiokutsut.

Tällöin virtuaalikoneen pinossa on kehysosoittimen osoittamassa paikassa vastaus lähetettyyn komentoon. Jos vastaus halutaan ottaa talteen, on lähdekoodissa kutsun perään laitettu lista muuttujista, joihin vastaus halutaan lukea. Kääntäjä on tällöin tehnyt tavukoodin, joka ohjaa virtuaalikoneen purkamaan vastausviestin näihin muuttujiin.

Lopuksi kun viesti on lähetetty ja vastaus käsitelty, virtuaalikone suorittaa seuraavan tavukoodin, joka on viestin lopetuskäsky. Tällöin palautetaan pino-osoitin osoittamaan viestiosoitinta ja palautetaan pinosta kehysosoitin ja viestiosoitin.

7 Esimerkki täsmäkielen käytöstä

Esimerkkisovelluksena käytetään Bluegigan ”Heart Rate Sensor” sovellusta (11). Sovellus toteuttaa Bluetooth Heart Rate profiilin (12). Sovelluksen tarkoituksena on näyttää, miten sykemittari eli sydämen sykkeen mittaus voidaan toteuttaa Bluegigan BLE113-moduulillla.

Sovellus lukee AD-mittausarvoa joka sekunti ja asettaa sen sopivasti skaalattuna GATT-tietokantaan, josta sen voi lukea Bluetooth 4.0:aa tukevalla laitteella, esimerkiksi matkapuhelimella.

Projektitiedosto

Projektitiedostossa määritellään XML-kielellä moduulin toiminta. Esimerkki listauksessa 4. Siinä määritellään, mille moduulille ollaan ohjelmisto tekemässä ja missä tiedostoissa tapahtuu tarkempi määrittely.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <script in="hr.bgs" />
    <image out="BLE113.hex" />
    <device type="ble113" />
    <boot fw="bootuart" />
</project>
```

Listaus 4. Projektitiedosto

GATT-tietokanta on määritelty tiedostossa ”gatt.xml”. Sitä ei käydä tässä läpi, mutta sen voi ajatella olevan hierarkinen tietokanta.

Kohdassa ”script”-kerrotaan, mistä löytyy BGScriptin lähdekoodi. Tässä tapauksessa se on tiedostossa ”hr.bgs”.

Esimerkin lähdekoodi käydään läpi seuraavaksi. Lähdekoodi on tekstitiedosto, joka voidaan kirjoittaa millä tahansa tekstieditorilla. Lähdekoodi on esitetty kokonaisuudessaan liitteessä 2.

Ohjelman alussa määritellään tarvittavat muuttujat. Tämä ohjelma käyttää kaksi tavuista tmp-taulukkoa ja 6-tavuista addr-taulukkoa.

```
#declare buffer for building attribute value
dim tmp(2)
dim addr(6)
```

Ohjelman käynnistyessä suoritetaan system_boot-tapahtuma, johon laitetaan kaikki moduulin alustukseen tarvittavat käskyt.

```
event system_boot(major,minor,patch,build,ll_version,protocol,hw)

    #Get local BT address
    call system_address_get( )(addr(0:6))

    # Write BT address to DI service serial number string
    call attributes_write(xgatt_dis_2a25,0,6,addr(0:6))

    #start advertising in connectable mode
    call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
    #timer at every second repeating
    call hardware_set_soft_timer(32768,0,0)
end
```

Tapahtuman käsittelyssä kutsutaan ensin system_address_get-käskyä, jonka paluuarvo laitetaan addr-taulukkoon. Käsky lukee moduulin Bluetooth-osoitteen. Tämä kirjoitetaan GATT-tietokannan ”xgatt_dis_2a25” nimiseen tietopaikkaan.

Seuraavaksi laitetaan moduuli mainostamaan itseään kutsumalla gap_set_mode-käskyä.

Lopuksi laitetaan ajastin päälle hardware_set_soft_timer-käskyllä. Ajastin määritellään antamaan tapahtuma joka sekunti.

```
event hardware_soft_timer(handle)
    #measure potentiometer
    call hardware_adc_read(6,1,2)
end
```

Hardware_soft_timer-tapahtumaa kutsutaan ajastimen toimesta. Tapahtumassa luetaan hardware_adc_read-käskyllä AD-muuntimen arvo. Arvoa ei saada paluuarvona, vaan siitä tulee erillinen hardware_adc_result-tapahtuma kun AD-muuntimen mittaus on valmis.

```
event hardware_adc_result(input,value)
    #build simple characteristic value response
    tmp(0:1)=2
    #calculate some valid hr value 20-224
    tmp(1:1)=value/160+20

    call attributes_write(xgatt_hrs_2a37,0,2,tmp(0:2))
end
```

Hardware_adc_result-tapahtumassa määritellään tmp-taulukkoon Heart Rate-spesifikaation mukaiset arvot ja kirjoitetaan se ”xgatt_hrs_2a37” nimiseen tietopaikkaan GATT-tietokantaan. Tietokannasta päätelaite voi sen edelleen lukea.

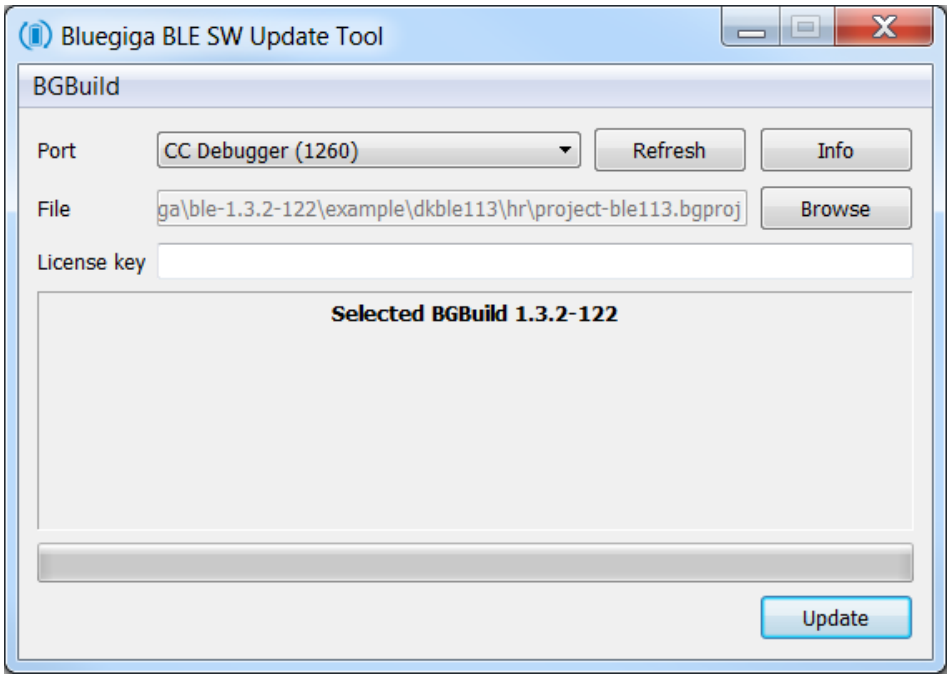
Connection_disconnected-tapahtuma saadaan, kun ulkopuolinen yhteys on katkennut.

```
event connection_disconnected(handle,result)
    #start advertising again after disconnection
    call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
end
```

Koska moduuli lopettaa mainostamisen, kun yhteys avataan, niin sulkiessa yhteys se joudutaan aloittamaan uudelleen. Tämä tehdään jälleen gap_set_mode-käskyllä.

Ohjelman kääntämisen ja moduulin lataamisen helpottamiseksi on toteutettu työkalu tämän projektin ulkopuolella. Työkalu on esitetty kuvassa 12.

Työkalussa valitaan projektitiedosto ja painamalla update-näppäintä se automaattisesti kääntää lähdekoodin ja ohjelmoi sen moduuliin.



Kuva 12. BLE-Moduulien ohjelmointityökalu.

Kun Bluegigan ohjelmisto sekä käännetty BGScript on ladattu moduuliin, ohjelmisto käynnistyy automaattisesti ja alkaa suorittaa tehtyä ohjelmakoodia. Ohjelmiston toimivuuden toteamiseen voidaan käyttää esimerkiksi matkapuhelimen sovellusta.

8 Yhteenveto

BGScript on otettu ensimmäisenä käyttöön Bluegigan Bluetooth low energy tuotteissa, näistä asiakkaista sitä käyttää noin 90 %. Kokonaisuudessa se on käytössä jo yli miljoonassa myydyssä moduulissa. Asiakkaat ovat myös hyvin sisäistäneet, että kielen tarkoituksena on automatisoida yksinkertaisia toimintoja, joihin on aikaisemmin vaadittu ulkopuolista prosessoria.

Ongelmina on BGScriptissä muodostunut, että kielessä ei ole itsessään minkäänlaista tukea asiakkaan ohjelmiston testaamiselle. Asiakas joutuu tällöin tulostusviesteillä ja ledien vilkuttamisilla yrittää selvittää ohjelmansa toimintaa. Myös on kaivattu erilaisia kirjastofunktioita, joissa olisi valmiina erilaisia tekstinkäsittelyfunktioita.

Alkuperäinen ajatus kielestäe oli, että ohjelmat olisivat enintään muutamia kymmeniä rivejä, joissa esimerkiksi luetaan lämpömittarin lukema SPI-väylän kautta ja kirjoitetaan se sopivasti muokattuna Bluetooth low energyn GATT-kantaan etälaitteen luettavaksi. Asiakkaat ovat kuitenkin onnistuneet tekemään tuhansien rivien ohjelmia ja ovat muun muassa vaatineet BGScriptiin tuen useammalle lähdekooditiedostolle.

Myös kilpailijat ovat huomanneet, kuinka hyödyllistä on antaa asiakkaan ohjelmoida itse toiminnallisuutta moduuliin, ja he tarjoavatkin nykyään vastaavanlaisia täsmäkieliä. Näin ollen tämän kielen kehittäminen on moduulialalla ollut eräänlainen ajattelutavan muutos, joka on pakottanut kaikki toimijat toimenpiteisiin pysyäkseen kilpailussa mukana.

Asioiden Internet (Internet of Things) on yksilöitävissä olevien sulautettujen järjestelmien yhteenliittymä Internetissä. Esimerkiksi Konecranesin toimitusjohtaja Pekka Lundmark uskoo sen olevan seuraava suuri teollinen vallankumous (13). Bluegigan langattomien moduulien toteuttama BGScript-kieli tarjoaa valmiudet jokaiselle laitevalmistajalle asioiden Internetiin.

Lähteet

1 Products. 2014. Verkkodokumentti. Bluegiga Technologies Oy. <https://www.bluegiga.com/en-US/products/>. Luettu 10. 3 2015.

2 Bluegiga Bluetooth Smart Software, V.1.3 API Documentation. 2014. Bluegiga Technologies Oy.

3 Haltsonen, Seppo ja Rautanen, Esko T. 2008. Tietokonetekniikka. Helsinki: Edita.

4 BGLIB workshop meeting. Pöytäkirja. 10.08.2010. Bluegiga Techonologies Oy

5 Forth Interest Group. 2013. Verkkodokumentti. <http://www.forth.org>. Päivitetty 15.3.2013. Luettu 10.3.2015

6 eLUA - eluaproject. 2011. Verkkodokumentti. <http://www.eluaproject.net/>. Luettu 10.3.2015

7 Tcl Developer Site. 2015. Verkkodokumentti. <http://www.tcl.tk/>. Luettu 10.3.2015

8 Kemeny, John G. ja Kurtz, Thomas E. 1986. BASIC, Fourth Edition. Dartmouth: Dartmouth College.

9 Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D. 2007. Compilers: Principles, Techiques & Tools. Boston: Addison-Wesley.

10 Dijkstra, E.W. 1961. ALGOL-60 Translation. Amsterdam: Mathematisch centrum.

11 Bluetooth® 4.0 Heart rate sensor. 2014. Bluegiga Technologies Oy.

12 Heart rate profile. 2011. Bluetooth SIG.

13 Älyn lisääminen konetuotteisiin johtaa teolliseen vallankumoukseen. 31. 8. 2013. Helsingin Sanomat. Verkkodokumentti. <http://www.hs.fi/talous/a1377840634455>. Luettu 3.12.2014.

BGScript™-käskykanta

Tavu	Käsky	kuvaus
0x1B	32BIT	Laita seuraavat 4 tavua pinoon
0x16	32BIT8	Laita seuraava tavu pinoon 4 tavun lukuna
0x01	16BIT	Laita seuraavat kaksi tavua pinoon
0x10	8BIT	Laita seuraava tavu pinoon
0x02	SUM	Yhteenlasku
0x03	DEC	Vähenneslasku
0x04	AND	Binäärinen ja
0x05	OR	Binäärinen tai
0x06	XOR	Binäärinen ehdoton ei
0x42	SHFTL	Siirrä lukua biteittäin vasemmalle
0x43	SHFTR	Siirrä lukua biteittäin oikealle
0x40	LAND	Looginen ja
0x41	LOR	Looginen tai
0x07	MUL	Kertolasku
0x08	DIV	Jakolasku
0x09	EQ	Vertaa lukujen yhtäsuuruutta
0x0A	LT	Vertaa onko ensimmäinen luku pienempi
0x32	GT	Vertaa onko ensimmäinen luku suurempi
0x44	LTE	Vertaa onko ensimmäinen luku pienempi tai yhtäsuuri
0x45	GTE	Vertaa onko ensimmäinen luku suurempi tai yhtäsuuri
0x32	NE	Vertaa ovatko luvut erisuuria
0x0B	IF	Jos luku erisuuri kuin 0, hyppää osoitteeseen
0x0C	READ	Lue luvun osoittamasta muistipaikasta luku

Tavu	Käsky	kuvaus
0x0D	READBUF	Lue muistista taulukko pinoon
0x0E	WRITE	Kirjoita luku muistipaikkaan
0x0F	WRITEBUF	Kirjoita taulukko muistiin
0x1C	FRAME	Lue luku kehysmuistista
0x1D	FRAMEBUF	Lue taulukko kehysmuistista
0x1E	FRAMESIGN	Lue luku kehysmuistista säilyttäen etumerkin
0x14	GOTO	Hyppää osoitteeseen
0x15	MSGBEGIN	Aloita BGAPI viesti
0x16	MSGCALL	Lähetä BGAPI viesti
0x2F	MSGFINISH	Lopeta BGAPI viesti
0x17	END	Lopeta suoritus
0x18	ISTRING	Lue seuraava taulukko pinoon
0x19	DROP3	Poista 3 tavua pinosta
0x1A	DROP2	Poista 2 tavua pinosta
0x23	DROP1	Poista tavu pinosta
0x21	SWAP	Vaihda pinon kaksi päällimmäistä lukua
0x24	SFLOAT	Muuta luku IEEE SFLOAT muotoon
0x25	FLOAT	Muuta luku IEEE FLOAT muotoon
0x30	POPF	Siirrä kehyksestä luku muistiin
0x31	POPBUF	Siirrä kehyksestä taulukko muistiin
0x34	MEMCPY	Kopioi taulukko muistissa
0x35	MEMSET	Aseta taulukko muistissa

Esimerkkisovelluksen lähdekoodi

```
#declare buffer for building attribute value
dim tmp(2)
dim addr(6)

event system_boot(major,minor,patch,build,ll_version,protocol,hw)

    #Get local BT address
    call system_address_get( )(addr(0:6))

    # Write BT address to DI service serial number string
    call attributes_write(xgatt_dis_2a25,0,6,addr(0:5))

    #start advertising in connectable mode
    call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
    #timer at every second repeating
    call hardware_set_soft_timer(32768,0,0)
end

event hardware_soft_timer(handle)
    #measure potentiometer
    call hardware_adc_read(6,1,2)
end

event hardware_adc_result(input,value)
    #build simple characteristic value response
    tmp(0:1)=2
    #calculate some valid hr value 20-224
    tmp(1:1)=value/160+20

    call attributes_write(xgatt_hrs_2a37,0,2,tmp(0:2))
end

event connection_disconnected(handle,result)
    #start advertising again after disconnection
    call gap_set_mode(gap_general_discoverable,gap_undirected_connectable)
end
```