VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tuomas Holma

BLUETOOTH MESH-TEKNOLOGIAN SOVELTAMINEN TERMOSTAATTI-JÄRJESTELMÄSSÄ

Tekniikka
2021

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Tuomas Holma
Opinnäytetyön nimi	Bluetooth mesh-teknologian soveltaminen termostaattijärjestelmässä
Vuosi	2021
Kieli	suomi
Sivumäärä	52 + 4 liitettä
Ohjaaja	Jani Ahvonen

Tässä opinnäytetyössä tutustuttiin Bluetooth Mesh-teknologiaan ja sen käyttömahdollisuuksiin älykkään termostaattijärjestelmän kehityksessä. Työn tarkoituksena oli tutkia, millä tavalla Bluetooth Mesh-teknologiaa voidaan hyödyntää kotiautomaatiotarkoitukseen ja rakentaa lopullisen tuotteen kehitystä helpottava malli termostaattijärjestelmästä kehitysalustojen avulla.

Termostaattijärjestelmämallin tärkeimmät osat ovat lämpötilan mittauslaite ja verkkoa ylläpitävä laite. Järjestelmässä käytetään Silicon Labsin kehitysalustoja, ohjelmointiympäristöä sekä ohjelmistokehityspakettia. Järjestelmän provisiointiin käytetään Silicon Labsin Bluetooth mesh-puhelinsovellusta. Lopuksi mittauslaitteen koodi siirretään kehitysalustalta prototyyppilevylle.

Opinnäytetyö tuo yritykseen lisää Bluetooth-osaamista ja helpottaa tuotteiden kehitystä. Mallijärjestelmän kehittämisen aikana ilmeni myös jatkokehitysideoita, joita voidaan hyödyntää tulevaisuudessa.

Avainsanat: Bluetooth Mesh, termostaattijärjestelmä, sulautetut järjestelmät

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Tuomas Holma
Title	Application of Bluetooth Mesh-Technology in a Thermostat System
Year	2021
Language	Finnish
Pages	52 + 4 appendices
Name of Supervisor	Jani Ahvonen

The purpose of this thesis was to evaluate the application of the Bluetooth mesh-technology in the development of a smart thermostat system. The project includes research about the Bluetooth mesh technology and the use of evaluation boards in a model thermostat system. The model system is used to help with the development of a real thermostat system in the future.

The most important parts of the system are the temperature measuring device and the device that maintains the mesh-network. The evaluation boards, development environment, software development kit and the Bluetooth mesh-phone application used in this project are developed by Silicon Labs. The code of the measuring device was also be ported to a prototype board.

This thesis aims to bring more Bluetooth knowledge to the company and help with product development. Ideas for further development came up during the project and they can be applied to future devices.

Keywords: Bluetooth mesh, thermostat and embedded systems

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

LYHENTEET JA TERMIT

- BLE — Bluetooth Low Energy, likiverkkoteknologia
- GATT — General Attribute Profile, Määrittelee tiedonsiirtotavan BLE-laitteissa.
- Provisionti — Laitteen lisääminen mesh-verkkoon
- LPN — Low Power Node, Matalavirtainen mesh-laite
- Netkey — Mesh-verkon avain
- Appkey — Sovelluskohtainen avain mesh-verkossa
- DevKey — Laitekohtainen avain mesh-verkossa
- Friend — Laite, joka ylläpitää verkkoa LPN-laitteille
- managed flood — Mesh-verkon viestienvälitystekniikka
- I2C — Tiedonsiirtoprotokolla
- Kehitysalusta — Mikroprosessorin tai muun laitteen ympärille rakennettu piirilevy, joka sisältää tarvittavat oheislaitteet ja komponentit toimiakseen. Kehitysalustoja käytetään prototyyppien luomiseen ja mikro-ohjainten ohjelmoinnin opetteluun.
- WSTK mainboard — Wireless Starter Kit Mainboard, Silicon Labsin valmistama kehitysalusta.
- AEM — Advanced Energy Monitor, WSTK mainboard-kehitysalustan ominaisuus, jolla voidaan seurata virrankulutusta.
- PTI — Packet Trace Interface, WSTK mainboard-kehitysalustan ominaisuus, jolla voidaan seurata Bluetooth-pakettien liikennettä.
- switch-case — Ohjelmistorakenne, jossa ohjelma ohjataan tiettyyn haaraan muuttuvan arvon perusteella.

LIITELUETTELO

1 JOHDANTO

Tämän työn tarkoitus oli suunnitella ja toteuttaa malli Bluetooth Mesh-teknologiaa käyttävästä termostaattijärjestelmästä, joka toimii apuna lopullisen tuotteen kehitykselle. Järjestelmän tulee pystyä toimimaan omakotitaloympäristössä ja lämpötilan mittauslaitteiden tehontarpeen tulisi mahdollisimman pieni, jotta ne toimisivat paristolla mahdollisimman kauan.

Koska lämpötilan mittauslaitteiden on oltava matalatehoisia, ne eivät voi pitää Bluetooth Mesh-verkkoa yllä itsekseen. Tämän takia tarvitaan myös verkkovirrassa kiinni olevia laitteita, jotka pitävät verkkoa yllä ja välittävät lämpötilatietoa eteenpäin.

Termostaattijärjestelmän mallissa verkkoa ylläpitävinä laitteina käytettiin Silicon Labs:n WSTK mainboard-kehitysalustoja, joihin kytkettiin EFR32MG21-radiokortit. Lämpötilan mittauslaitteina käytettiin Thunderboard EFR32-kehitysalustoja, jotka sisältävät si7010-lämpötila-anturin. Mittauslaitteen ohjelma siirrettiin myös kustomoidulle prototyyppilevylle. Laitteet liitettiin mesh-verkkoon käyttämällä Silicon labs:n Bluetooth Mesh-puhelinsovellusta. Kehitysalustojen ohjelmointi tapahtui C-kielellä Silicon Labs:n Bluetooth Mesh-ohjelmistokehityspakettia käyttäen. Tässä opinnäytetyössä käydään läpi Bluetooth Mesh-teknologian teoriaa ja sen soveltamista termostaattijärjestelmän kehitykseen mallijärjestelmän avulla.

Tämän työn toimeksiantajana toimi Seinäjoella sijaitseva Xedi Ky, joka on asiakasräätälöityihin sulautettuihin järjestelmiin erikoistunut yritys. Opinnäytetyö sai alkunsa asiakasyrityksen tilaamasta projektista. Xedi Ky on perustettu vuonna 2019.

2 BLUETOOTH MESH-TEKNOLOGIA

Tässä osiossa käydään läpi Bluetooth mesh-standardissa määriteltyä yleistä tietoa Bluetooth mesh-verkon teoriasta ja toiminnasta.

2.1 Yleistä

Bluetooth Low Energy on langaton tiedonsiirtotekniikka, jota on käytetty vuodesta 2010 asti yhteisenä kommunikointitapana matalatehoisille laitteille teollisuudessa ja kuluttajatuotteissa. Vuonna 2017 julkaistiin uusi Bluetooth mesh-standardi, joka pohjautuu BLE-teknologiaan. Mesh-standardi mahdollistaa perinteisten Bluetooth-laitteiden kommunikoinnin mesh-verkossa, joka avaa laitteille paljon uusia ominaisuuksia ja käyttötapoja. BLE ja Bluetooth mesh käyttävät samaa 2,4 GHz taajuusaluetta.

Bluetooth-laitteet kommunikoivat yleisimmin yhdeltä-yhdelle tai yhdeltä-monelle yhteydellä. Tämä toimii hyvin esimerkiksi älypuhelimissa, joihin on mahdollista parittaa samanaikaisesti monta laitetta. Mesh-verkon etu normaaleihin Bluetooth-verkkoihin on monelta-monelle yhteys, jonka avulla kaikki verkossa olevat laitteet pystyvät kommunikoimaan toistensa kanssa. Monelta-monelle-yhteyden avulla voidaan luoda Bluetooth-verkkoja suuremmalla laitemäärällä ja laajemmalla käyttöalueella. Verkoista voidaan myös tehdä joustavampia ja varmempia, sillä yksittäisten laitteiden häiriöt ja yhteyksien menetykset eivät välttämättä kaada koko verkkoa /2/.

2.2 Bluetooth mesh-käsitteet

Bluetooth mesh-standardi sisältää useita käsitteitä, jotka on ymmärrettävä ennen teknologian hyödyntämistä järjestelmässä. Oleellisimmat käsitteet ovat mesh-verkon provisiointi, viestit, osoitteet, julkaisu/tilaus, tilat, silmukan ominaisuudet sekä mesh-verkon arkkitehtuuri. Näistä käsitteistä monet ovat Bluetooth mesh-keskeisiä ja ne eivät esiinny normaalissa Bluetooth/BLE kehityksessä.

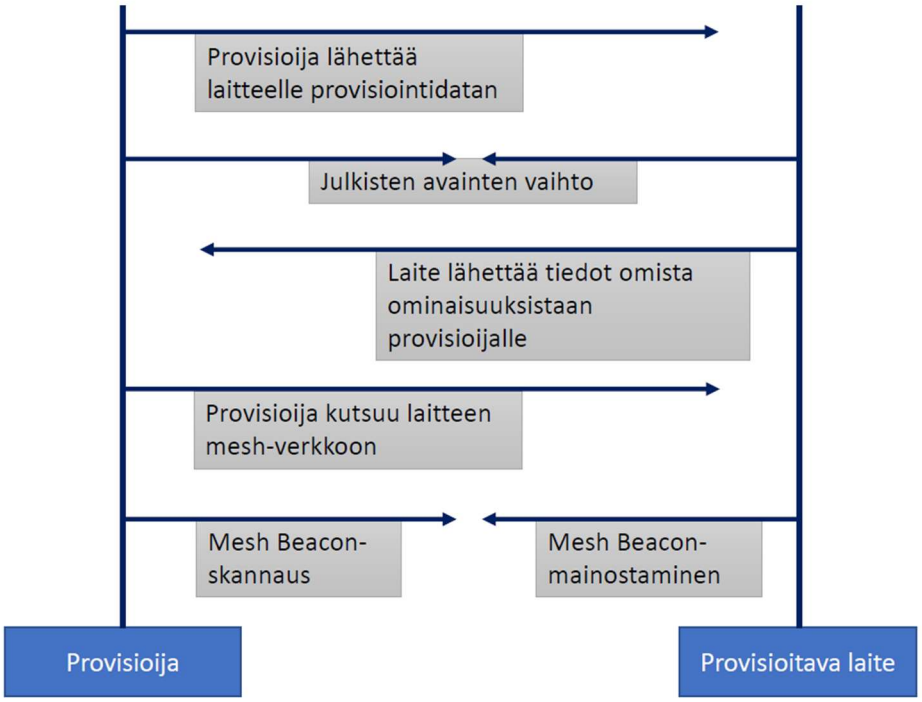
2.2.1 Mesh-verkon provisiointi

Provisiointi tarkoittaa uuden laitteen lisäämistä mesh-verkkoon. Laitteiden provisiointia varten tarvitaan provisioija, joka voi olla esimerkiksi älypuhelin tai muu kannettava laite. Provisioinnin tarkoituksena on saada tiedot uuden laitteen ominaisuuksista, antaa laitteelle verkkotiedot ja luoda turvallinen yhteys salausavaimien avulla. Mesh-verkkoon provisioituja laitteita kutsutaan silmukoiksi.

Provisiointi alkaa sillä, että uusi provisioitava laite mainostaa itseään normaalin Bluetooth-laitteen tapaan, mutta normaalien mainostuspakettien sijaan laite lähettää Mesh Beacon-mainostuspaketteja. Provisioija tunnistaa uuden laitteen mainostuspakettien avulla ja lähettää laitteelle kutsun mesh-verkkoon. Kutsun saatuaan provisioitava laite lähettää provisioijalle tiedot omista ominaisuuksistaan. Tämä vaihe on samankaltainen kuin normaalien Bluetooth-laitteiden paritus /7/.

Kun laite on kutsuttu verkkoon ja provisioija on saanut tiedot laitteen ominaisuuksista, laite ja provisioija vaihtavat keskenään julkiset avaimensa. Uusi laite on myös varmennettava käyttäjän toimesta, joten provisioija ei voi lisätä laitteita turvallisesti verkkoon ilman käyttäjää. Laitteen varmennuksen jälkeen provisioija lähettää uudelle laitteelle provisiointidatan. Provisiointidata sisältää NetKeyn, IV-indexin sekä unicast-osoitteen. Kuva 1 havainnollistaa provisiointiprosessia.

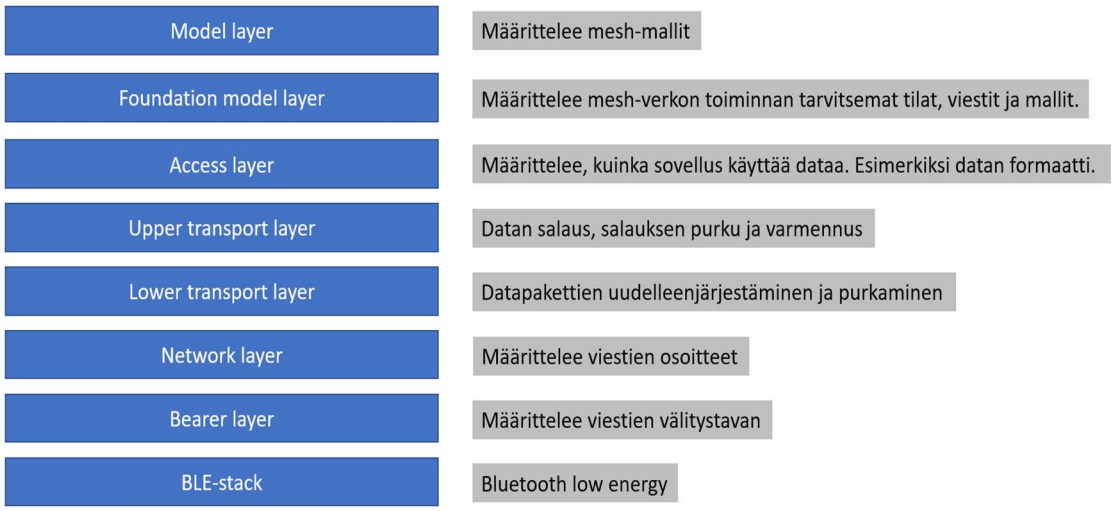
NetKey on jaettu avain mesh-verkolle, jonka kaikki verkossa olevat laitteet tietävät. Provisiontivaiheessa laitteelle jaetaan usein myös AppKey-avain, jonka laite tarvitsee sovelluskohtaista dataa purkaakseen. Verkko voi sisältää useita AppKey-avaimia, sillä verkossa voi toimia monia erilaisia sovelluksia. Jokaisella laitteella on myös oma DevKey eli laitekohtainen avain. IV-indexi on laitteiden yhteinen turvallisuusparametri /2/.



Kuva 1. Uuden laitteen provisionti mesh-verkkoon

2.2.2 Mesh-verkon arkkitehtuuri

Mesh-arkkitehtuurille on määritelty seitsemän protokollakerrosta. Kaikilla kerroksilla on omat tehtävät mesh-verkon toiminnan toteutumiseksi ja ylemmät kerrokset ovat riippuvaisia alempien kerrosten toiminnasta. Bluetooth mesh toimii Bluetooth low energy-standardin päällä, joten BLE-pino on alimpana kerroksena mesh-arkkitehtuurissa. Bluetooth mesh-protokollapino on havainnollistettuna kuvassa 2 /4/.



Kuva 2. Mesh-verkon protokollapino

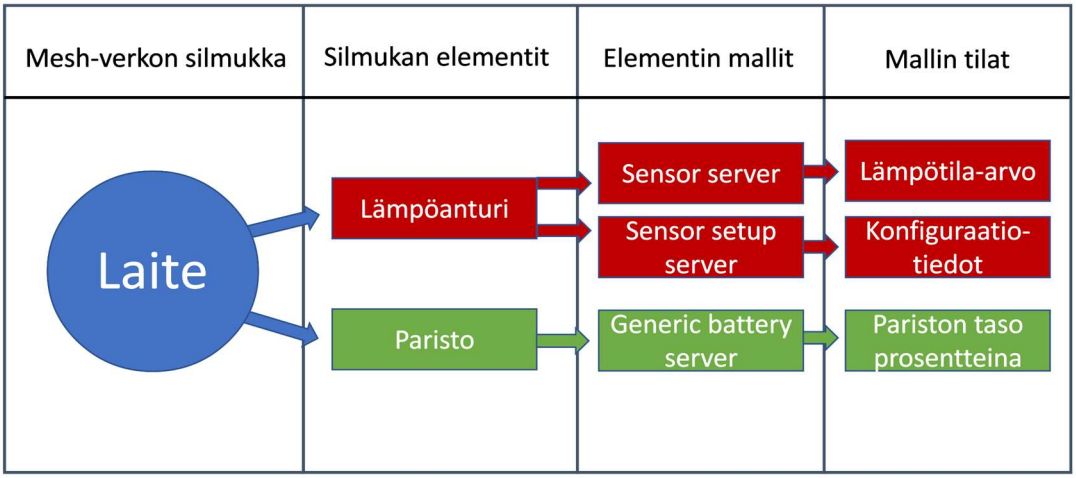
2.2.3 Laitteiden kommunikointi mesh-verkossa

Laitteiden kommunikointi mesh-verkossa perustuu viesteihin. Mesh-verkoissa ei ole normaalia reititystä, vaan ne käyttävät managed flooding-tekniikaksi kutsuttua tapaa viestien välitykseen. Tämä tarkoittaa sitä, että laitteen lähettämät viestit lähetetään kaikille sen kantoalueella oleville laitteille. Ruuhkautumisen välttämiseksi mesh-teknologiassa hyödynnetään viestien time to live-arvoa (TTL) sekä laitteiden välimuistia, jonka avulla samaa viestiä ei lähetetä monta kertaa.

Laitteiden kommunikaatio toimii julkaisu/tilaus periaatteella, jossa julkaisu tarkoittaa viestin lähettämistä. Tilauksella tarkoitetaan sitä, että laitteet voidaan konfiguroida lukemaan tietyistä osoitteista julkaistuja viestejä. Laite voi tilata viestejä yksittäisen laitteen osoitteesta tai ryhmältä. Mesh-verkkoon voidaan luoda usean laitteen ryhmiä, joilla on yksi ryhmäosoite. Ryhmäosoitteen alle voidaan sisällyttää esimerkiksi yhden huoneiston termostaatit, jolloin huoneiston ohjauslaite voi tilata viestejä vain sille kuuluvilta laitteilta /1/, /2/.

Viestejä voidaan lähettää mesh-verkossa kahden erilaisen kantajan avulla. Advertising bearer eli mainostuskantaja lähettää ja vastaanottaa mesh-viestejä Bluetooth-mainostusominaisuuden avulla. General Attribute eli GATT-kantaja mahdollistaa normaalien Bluetooth-laitteiden kommunikoinnin mesh-verkossa proxy-protokollan avulla /3/.

Yksittäisen silmukan sisällä voi olla useita mesh-verkon avulla hallittavia osia, joita kutsutaan elementeiksi. Elementeillä on omat osoitteensa ja niiden lähettämä tai vastaanottama data määritellään mallien ja tilojen avulla. Tilat määrittelevät, minkälaista dataa elementti käsittelee. Mallit ovat valmiita tilakokoelmia, jotka ovat määriteltyjä mesh-standardissa. Mallien avulla kehitys helpottuu, sillä esimerkiksi antureille voidaan useimmiten käyttää tarvittavat tilat sisältävää ja valmiiksi määriteltyä sensor-mallia. Kuvassa 3 on suunnitelma termostaattijärjestelmän mittauslaitteen elementeistä, malleista ja tiloista /1/.



Kuva 3. Langattomalle anturille suunnitellut elementit, mallit ja tilat

2.2.4 Mesh-silmukoiden ominaisuudet

Provisioitujen laitteiden eli silmukoiden perusominaisuus on viestien lähettäminen. Silmukoille voidaan antaa myös erityisominaisuuksia, jotka laajentavat niiden toiminnallisuutta. Erityisominaisuuksiin kuuluvat proxy-, relay-, low power- sekä friend-ominaisuudet. Nämä ominaisuudet ovat kaikki olennaisia termostaattijärjestelmän toiminnassa.

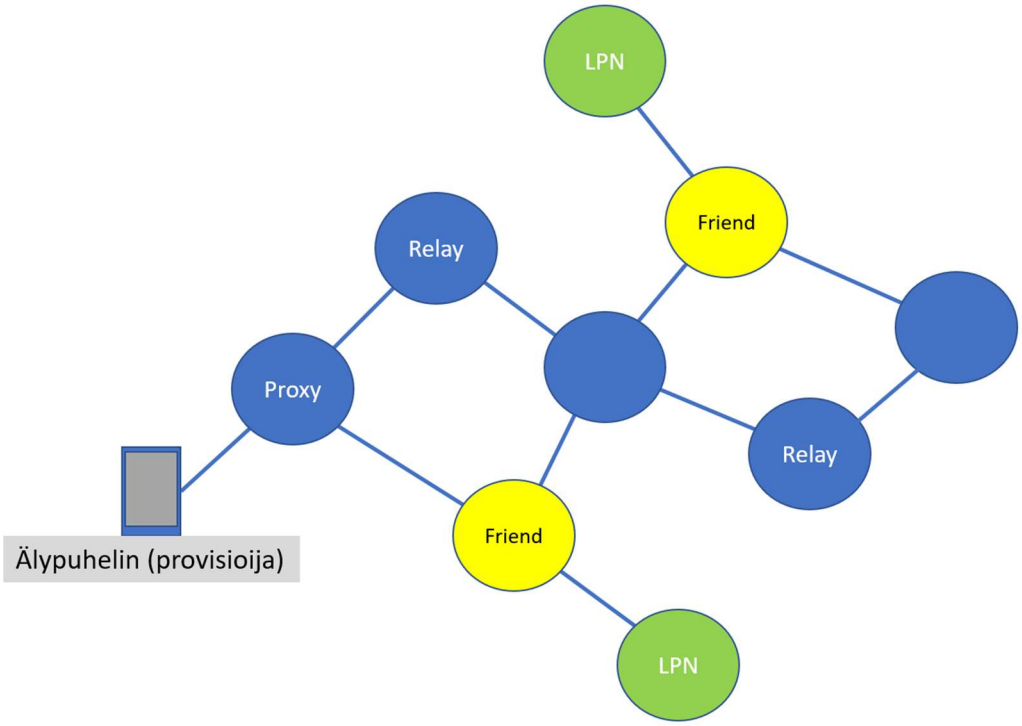
Proxy-ominaisuus mahdollistaa BLE-laitteiden kommunikoinnin mesh-verkossa, vaikka laitteissa ei ole suoranaista Bluetooth mesh-tukea. Tällaisia laitteita ovat yleisimmin perinteiset BLE-laitteet. Proxy-silmukat pystyvät välittämään viestejä GATT- ja mainostuskantajien välillä, joten ne toimivat verkon sisäänpääsykohtana tällaisille laitteille.

Relay-ominaisuuden avulla silmukat pystyvät välittämään viestejä eteenpäin, jolloin mesh-verkon käyttöalue kasvaa. Tämä ominaisuus on yleensä käytössä vain verkkovirrassa kiinni olevissa laitteissa, sillä se vaatii paljon virtaa toimiakseen. Relay-ominaisuus välittää vain mainostuskantajan avulla lähetettyjä viestejä.

Low power-ominaisuuden ansiosta mesh-verkkoon voidaan lisätä paristokäyttöisiä laitteita. Low power node (LPN) on laite, joka on suurimman osan ajasta pienimmässä mahdollisessa energiatilassa säästääkseen virtaa. Laite herää määritellyin aikavälein, jolloin se ottaa yhteyden friend-silmukkaan. Yhteyden aikana LPN voi kommunikoida mesh-verkon muiden laitteiden kanssa friend-silmukan kautta.

Friend-silmukka on verkkovirrassa kiinni oleva laite, joka kerää muistiinsa LPN-laitteelle tarkoitettuja viestejä. LPN-laitteen herätessä unitilasta friend-silmukka välittää kerätyt viestit laitteelle tai vastaanottaa LPN-laitteelta dataa.

Low power- ja friend-ominaisuudet ovat keskeisessä osassa termostaattijärjestelmän kehitystä. Näiden ominaisuuksien avulla voidaan rakentaa paristoilla toimiva verkosto antureista, jotka pystyvät kommunikoimaan järjestelmän muiden laitteiden kanssa matalasta virrankulutuksesta huolimatta. Kuvassa 4 on esimerkkitopologia mesh-verkosta, joka sisältää LPN- ja friend-laitteet sekä provisioijan /5/, /1/.



Kuva 4. Mesh-esimerkkitopologia

3 TERMOSTAATTIJÄRJESTELMÄMALLIN VAATIMUSTEN MÄÄRITTELY

3.1 Käyttötarkoitus

Järjestelmän tarkoitus on mitata yhden tai useamman huoneiston lämpötilaa ja mahdollistaa lämpötilatiedon välittäminen ohjauslaitteille langattomasti. Järjestelmän tulee soveltua omakotitaloympäristöön.

3.2 Toiminnalliset vaatimukset

3.2.1 Lämpötilan mittaaminen

Antureiden tulee pystyä mittaamaan ympäristön lämpötilaa 0,1 °C resoluutiolla ja ±0,5 °C tarkkuudella. Mittausalueen on oltava vähintään 0–50 °C.

3.2.2 Antureiden virrankulutus

Antureiden on pystyttävä toimimaan paristoilla yhtäjaksoisesti vähintään 6 kuukautta. Laitteen tulee pystyä mittaamaan tulojännite paristolta.

3.2.3 Lämpötilatietojen lähettäminen

Anturi lähettää lämpötilatietonsa verkkovirrassa olevalle ohjauslaitteelle tietyin väliajoin. Lämpötilatiedon lähetysintervallia on pystyttävä muuttamaan.

3.2.4 Laitteiden kommunikointi

Järjestelmän Bluetooth-laitteiden kommunikaation on toimittava Bluetooth-standardin mukaisesti. Verkkovirrassa olevien laitteiden tulee pitää mesh-verkko yllä jatkuvasti ja paristokäyttöisten laitteiden tulee olla lepotilassa aina, kun se on mahdollista.

3.3 Järjestelmän liittymät

3.3.1 Käyttöliittymä

Lopullisen järjestelmän käyttäjäympäristön tulee toimia ensisijaisesti puhelinsovelluksella. Mallijärjestelmässä laitteita seurataan tietokoneelta konsoliyhteyden avulla.

3.3.2 Laitteistoliittymät

Laitteissa on oltava riittävän tehokas Bluetooth-laitteisto mesh-verkon ylläpitämistä varten. Järjestelmän konfigurointi vaatii myös älypuhelimen, joka pystyy ajamaan Bluetooth mesh-sovellusta.

3.3.3 Ohjelmistoliittymät

Anturi on lepotilassa ja herää asetetuin aikavälein mittaamaan lämpötilatiedon. Lämpötilatieto lähetetään eteenpäin ohjauslaitteelle. Kommunikaatio tapahtuu Bluetooth mesh-standardin mukaisesti.

3.3.4 Tietoliikenneliittymät

Järjestelmän laitteita tulee pystyä hallitsemaan Bluetooth-yhteyden avulla.

3.4 Muut vaatimukset

3.4.1 Suorituskyky

- Mittausresoluutio 0.1 °C
- Mittausalue 0-50 °C
- Pariston kesto 6kk

3.4.2 Käytettävyys

Järjestelmän hallinnan tulee olla mahdollista kuluttajille ilman erityisosaamista.

3.4.3 Ylläpito

Antureita on pystyttävä poistamaan tai lisäämään puhelinsovelluksen avulla.

3.4.4 Ympäristö

Kaikki järjestelmän laitteet suunnitellaan sisäkäyttöön kiinteistöissä.

3.4.5 Turvallisuusvaatimukset

Laiteverkon tulee pysyä pystyssä, vaikka yksittäinen laite sammuisi. Laitteiden välinen kommunikaatio on salattava Bluetooth mesh-standardin mukaisesti.

3.5 Suunnittelun rajoitukset

3.5.1 Ohjelmiston rajoitukset

Ohjelmiston kehityksessä tukeudutaan Silicon Labs:n Bluetooth mesh-ohjelmistokehityspakettiin.

3.5.2 Laitteiston rajoitukset

Järjestelmän laitteiden on oltava tarpeeksi lähellä toisiaan, jotta mesh-verkko pysyy pystyssä.

4 TERMOSTAATTIJÄRJESTELMÄMALLIN SUUNNITTELU

4.1 Järjestelmän tarvittavat osat

Järjestelmä päätettiin rakentaa Silicon Labs:n järjestelmäpiirien ympärille, sillä valmistajan dokumentaatio Bluetooth mesh-teknologiasta todettiin riittävän kattavaksi. Piireissä on sisäänrakennettu Bluetooth-radio ja niiden ohjelmoimiseen voidaan käyttää Silicon Labs:n Bluetooth mesh-ohjelmistokehityspakettia. Kyseisiä piirejä oli käytetty myös aikaisemmin perinteisten Bluetooth-laitteiden kehityksessä.

Mallijärjestelmän kehittämisen tavoitteena on rakentaa toimiva mesh-verkko, jossa laiteyhteydet toimivat halutulla tavalla ja dataa pystytään lähettämään sekä lukemaan. Tällä tavalla lopullisen termostaattijärjestelmän kehityksessä voidaan ottaa huomioon mesh-kommunikaatiosta opitut asiat ja muihin ominaisuuksiin voidaan keskittyä enemmän.

Järjestelmämallin toteuttamiseen tarvitaan lämpötilan mittauslaite sekä Friend-laite. Mallijärjestelmässä Friend-laitteen tulisi pystyä ylläpitämään mesh-verkkoa nukkuville mittauslaitteille sekä jakamaan mittauslaitteiden lähettämä lämpötiladata muiden verkossa olevien laitteiden kanssa. Friend-ominaisuus voidaan siirtää lopullisessa järjestelmässä kaikkiin verkkovirrassa oleviin laitteisiin, joten järjestelmän mallissa Friend-laitteen voidaan kuvitella olevan esimerkiksi lämpötilan säätölaite tai ohjausyksikkö.

4.1.1 Low power-ominaisuus mittauslaitteessa

Mittauslaitteen virrankulutuksen minimoimiseksi mittauksia ei voida tehdä jatkuvasti, vaan niille on asetettava mittausintervalli. Tällä tavoin laite saa olla lepotilassa mittausten välillä. Mittauslaitteessa tullaan soveltamaan Low power-ominaisuutta, joka mahdollistaa laitteen kommunikoinnin mesh-verkossa matalavirtaisuudesta huolimatta.

4.2 Järjestelmän provisiointi

Mallijärjestelmän provisiointiin on helpointa käyttää Silicon Labs:n Bluetooth Mesh-sovellusta. Sovelluksen avulla on mahdollista yhdistää puhelin laitteeseen, jossa on proxy-ominaisuus. Tällä tavalla puhelin pystyy provisioimaan laitteita verkkoon sekä tarvittaessa muuttamaan niiden asetuksia ja poistamaan niitä verkosta.

4.3 Järjestelmän toiminta ja mesh-mallit

Järjestelmä tullaan rakentamaan Bluetooth Mesh-standardissa määriteltyjen sensor server- ja sensor client-mallien pohjalle. Sensor server-mallissa laite määritellään palvelimena, josta muut laitteet voivat pyytää antureiden dataa. Laitteet, jotka pyytävät palvelimelta dataa määritellään sensor client-mallilla.

Low power-ominaisuuden takia järjestelmän toimintamalli tulee kuitenkin olemaan hieman erilainen kuin standardissa määritelty malli. Mittauslaitteen on julkaistava lämpötiladata omatoimisesti herätessään. Friend-laite ohjelmoidaan odottamaan viestejä mittauslaiteilta, eikä se lähetä mittauslaitteelle pyyntöjä. Mittauslaitteiden julkaisuissa on oltava mukana lähettävän laitteen osoite, jotta Friend-laite tietää lämpötiladatan alkuperän monen laitteen kommunikoidessa sille samanaikaisesti. /8/.

5 TERMOSTAATTIJÄRJESTELMÄMALLIN RAKENTAMINEN

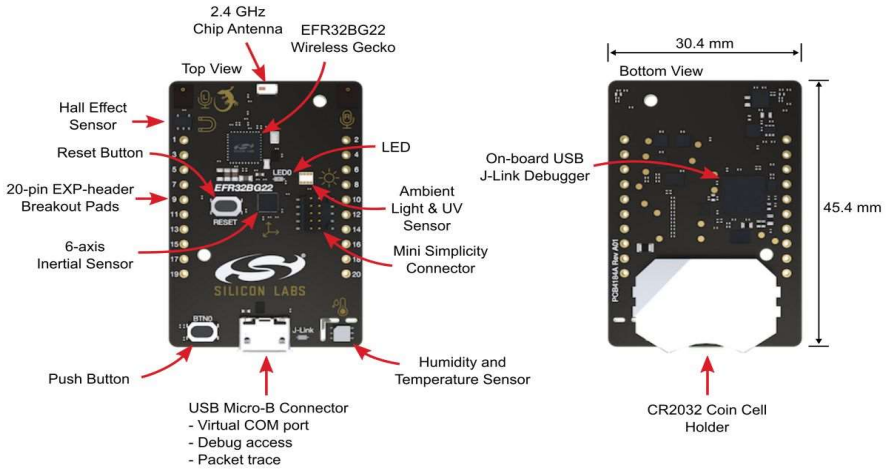
5.1 Käytetyt laitteet

Järjestelmä tulee sisältämään kaksi mittauslaitetta ja kaksi Friend-laitetta. Provisiointi tapahtuu Bluetooth mesh-puhelinsovelluksella ja laitteiden toiminnan seurantaan käytetään Tera Term-terminaaliemulaattoriohjelmaa. Lopuksi mittauslaitteen ohjelma siirretään kustomoidulle prototyyppilevylle.

5.1.1 Lämpötilan mittauslaite

Mittauslaitteen toimintaa evaluoidaan Thunderboard EFR32BG22-kehitysalustalla. Kehitysalusta on rakennettu EFR32BG22-järjestelmäpiirin ympärille ja se sisältää useita ympäristöantureita. Mittauslaitteen toiminnallisuus ohjelmoidaan kehitysalustalle, josta se voidaan myöhemmin siirtää prototyyppilaitteelle. Koodista on pyrittävä tekemään mahdollisimman siirrettävää, jotta siirtäminen prototyyppilaitteelle ei tuota suurta työmäärää. Liite 1 sisältää kehitysalustan tekniset tiedot.

Thunderboard EFR32BG22-kehitysalustassa on si7101-anturi, jolta mallijärjestelmä lukee lämpötilatiedon. Anturin tiedonsiirto kulkee I2C-väylän avulla. Ohjelman debuggaus ja sen lataaminen kehitysalustalle tapahtuu USB-yhteyden avulla. Kuvassa 5 on kehitysalustan layout-kuva /10/.

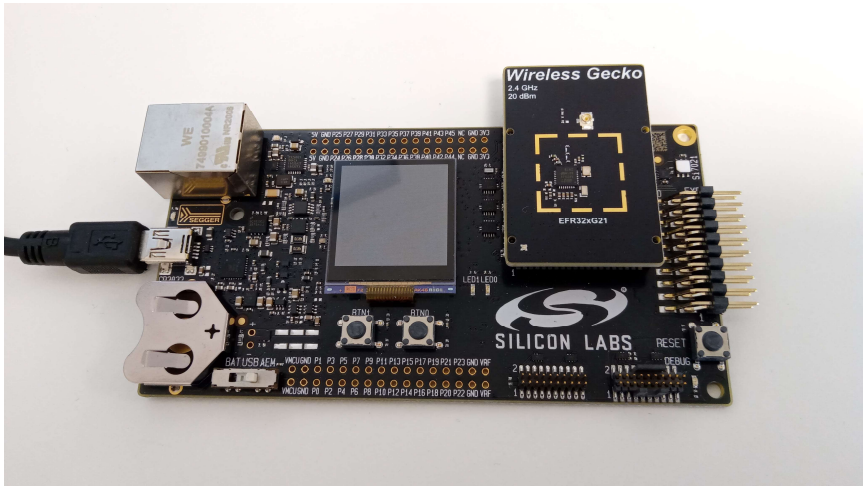


Kuva 5. Thunderboard EFR32BG22:n layout-kuva /10/

5.1.2 Friend-laite

Friend-laitteena järjestelmässä käytetään EFR32xG21-radiokorttia (**kuva 6.**), joka sisältää EFR32MG21-järjestelmäpiirin. Radiokortti on kytketty WSTK mainboard-kehitysalustaan, jonka avulla piiriä pystytään debuggaamaan. WSTK mainboard sisältää Advanced Energy Monitor (AEM)- ja Packet Trace Interface (PTI)-ominaisuudet, joita voidaan käyttää hyödyksi mesh-kehityksessä. Liite 2 sisältää laitteiden tiedot.

AEM mahdollistaa virrankulutuksen seurannan kytketystä laitteesta ja sitä tullaan käyttämään mittauslaitteen virrankulutuksen tutkinnassa. PTI:n avulla Bluetooth-pakettien liikennettä voidaan seurata ja tallentaa, joten sitä voidaan käyttää kommunikaatiovikojen etsintään /9/.



Kuva 6. WSTK mainboard ja EFR32xG21-radiokortti

5.1.3 Mittauslaitteen prototyyppilevy

Mittauslaitteen prototyyppi on kustomoitu piirilevy, jonka suunnitteli yrityksen elektroniikkasuunnittelija. Levy sisältää EFR32BG22-järjestelmäpiirin, MLX90614-lämpötila-anturin, Bluetooth-antennin sekä nastat debug-yhteyttä varten. Kuvassa 7 on kaksi prototyyppilevyä, joista toinen on kytketty WSTK mainboard-kehitysalustaan. Levy on suunniteltu siten, että ohjelman siirto kehitysalustalta olisi mahdollisimman vaivatonta.

Kuva 7. Prototyyppilevyt ja WSTK mainboard-kehitysalusta

5.1.4 Mesh-puhelinsovellus

Järjestelmän provisiointiin käytetään Silicon Labs:n Bluetooth Mesh-sovellusta. Sovelluksen avulla puhelimella voidaan provisioida laitteita mesh-verkkoon tai ottaa laitteeseen proxy-yhteys.

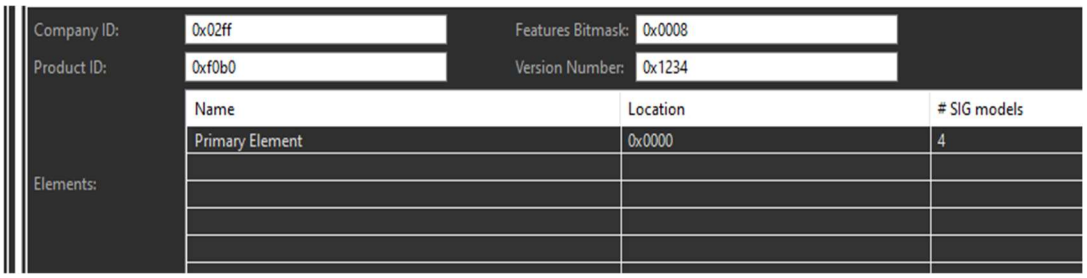
5.2 Laitteiden ohjelmointi

Laitteiden ohjelmointiin käytetään Simplicity Studio 4-ohjelmaa, joka on Silicon Labs:n kehittämä Eclipse-pohjainen ohjelmistokehitysympäristö. Ohjelmoinnissa tukeudutaan Bluetooth 2.17.3-ohjelmistokehityspakettiin sekä Silicon Labs:n esimerkkiohjelmiin, joita käytetään referensseinä ohjelmakokonaisuuden rakentamisessa ja Bluetooth-kommunikaation ohjelmoinnissa. AEM- ja PTI-ominaisuudet ovat myös käytettävissä Simplicity Studio 4:n sisällä. Referenssiohjelmina käytetään sensor server- ja sensor client-demoja.

Simplicity Studio luo Bluetooh mesh-projekteihin isc-tiedoston, jossa voidaan määritellä laitteen ominaisuudet. Ominaisuudet määritellään laskemalla halutuista ominaisuuksista bittimaski taulukon 1 mukaan. Mittauslaitteessa käytetään ainoastaan Low power-ominaisuutta, joten annetaan bittimaskiksi 1000. Friend-laitteen bittimaski on 0111, eli Low Power-ominaisuus on pois päältä ja kaikki muut ominaisuudet ovat päällä. Kuvassa 8 on esimerkkinä mittauslaitteen ominaisuuksien bittimaski /6/.

Bit	Feature	Notes
0	Relay	Relay feature support: 0 = False, 1 = True
1	Proxy	Proxy feature support: 0 = False, 1 = True
2	Friend	Friend feature support: 0 = False, 1 = True
3	Low Power	Low Power feature support: 0 = False, 1 = True
4–15	RFU	Reserved for Future Use

Taulukko 1. Laitteen ominaisuuksien bittimaskitaulukko /6/.



Kuva 8. Mittauslaitteelle asetettu bittimaski

5.2.1 Mittauslaitteen ohjelmointi

Mittauslaitteen ohjelmointi aloitetaan implementoimalla sensor server-malliohjelman mukainen switch-case-rakenne (**Kuva 9.**), jonka avulla laite osaa reagoida oikealla tavalla Bluetooth mesh-pinon tapahtumiin. Ohjelma on rakennettava pala palalta, sillä kehitysalusta ei ole yhteensopiva malliohjelman kanssa. Ohjelman tulee alustuksien jälkeen siirtyä pääsilmukkaan odottamaan tapahtumia, jotka ohjaavat ohjelmaa switch-case-rakenteen kautta.

```
    case gecko_evt_hardware_soft_timer_id:
      switch (pEvt->data.evt_hardware_soft_timer.handle) {
        case FACTORY_RESET_TIMER:
          // reset the device to finish factory reset
          gecko_cmd_system_reset(0);
          break;

        case RESTART_TIMER:
          // restart timer expires, reset the device
          gecko_cmd_system_reset(0);
          break;

        case NODE_CONFIGURED_TIMER:
        case FRIEND_FIND_TIMER:
          handle_lpn_timer_evt(pEvt);
          break;

        case SENSOR_DATA_TIMER:
          // send sensor data to friend device when timer expires
          handle_sensor_server_events(pEvt);
          break;

        default:
          break;
      }
      break;

    case gecko_evt_mesh_proxy_connected_id:
    case gecko_evt_mesh_proxy_disconnected_id:
      handle_mesh_proxy_events(pEvt);
      break;

    case gecko_evt_mesh_lpn_friendship_established_id:
    case gecko_evt_mesh_lpn_friendship_failed_id:
    case gecko_evt_mesh_lpn_friendship_terminated_id:
      handle_lpn_events(pEvt);
      break;

    default:
      //log("unhandled evt: %8.8x class %2.2x method %2.2x\r\n", evt_id, (ev
      break;
  }
}
```

Kuva 9. Laitteen toimintaa ohjaava switch-case-rakenne. Anturidatan lähetys tapahtuu, kun SENSOR_DATA_TIMER-ajastimen case-haaraan päädytään.

Low Power ominaisuuden mahdollistamiseksi luodaan lpn.c-tiedosto, jossa suoritetaan Low Power-ominaisuuden vaativat alustukset. Tiedostoon luodaan myös oma switch-case-rakenne (**Kuva 10.**). Ohjelma siirtyy tähän switch-case-rakenteeseen silloin, kun mittauslaitteen ja Friend-laitteen ystävyyssuhteen muodostaminen onnistuu, epäonnistuu tai jos ystävyyssuhde päättyy. Ystävyyssuhteen muodostamisen epäonnistuessa tai suhteen päättyessä ohjelma alkaa skannaamaan Friend-laitteita uudestaan. Jos ystävyyssuhde alkaa onnistuneesti, ohjelma aloittaa 10 sekunnin pituisen toistuvan ajastimen. Ajastimen alettua laite viedään unitilaan, jossa ainoastaan matalataajuiset kellot ovat päällä. Ohjelmaan tehdään myös toiminto, joka mittaa laitteen tulojännitteen pariston tason laskemista varten

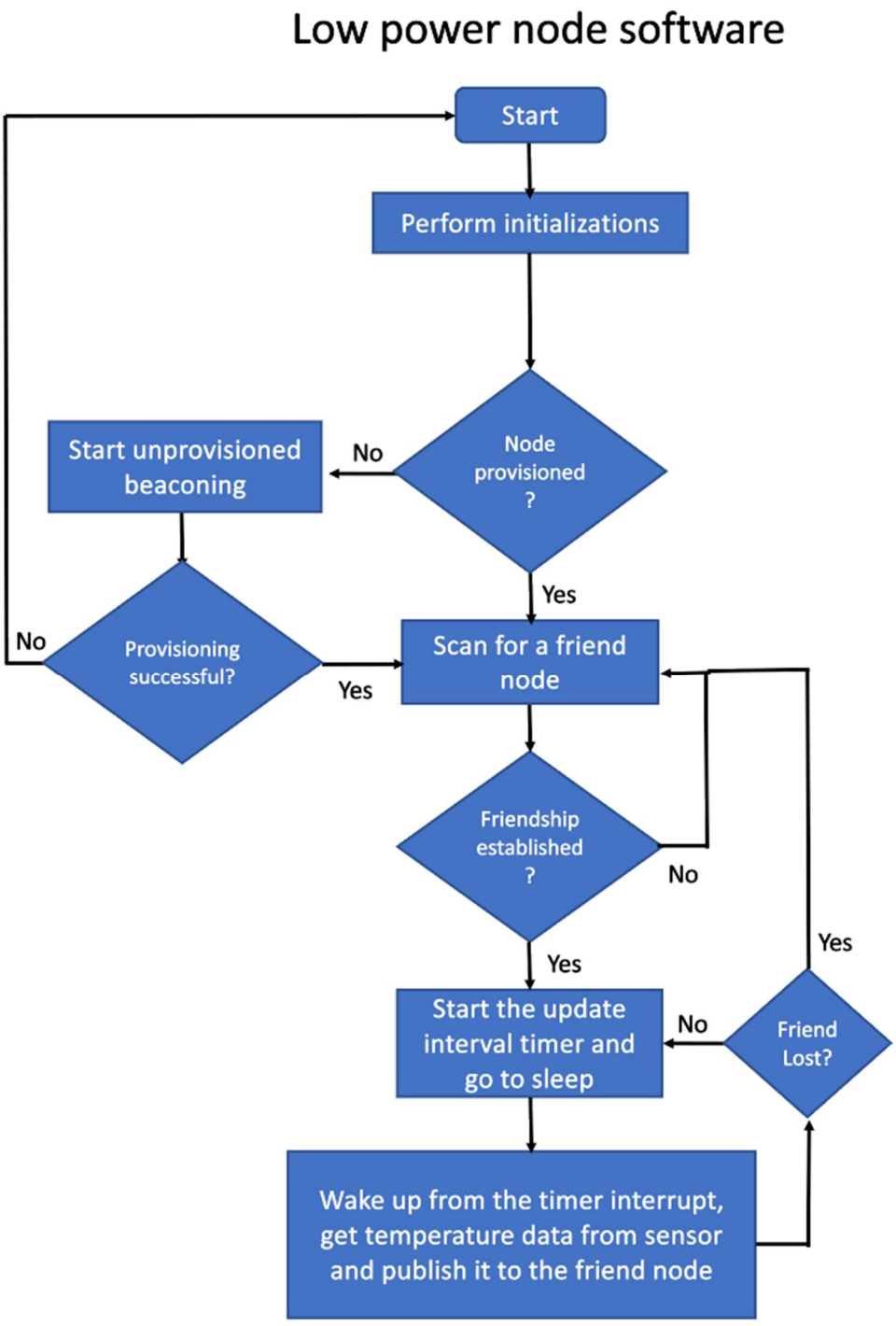
```
void handle_lpn_events(struct gecko_cmd_packet *pEvt)
{
  uint16_t result = 0;

  switch (BGLIB_MSG_ID(pEvt->header)) {
    case gecko_evt_mesh_lpn_friendship_established_id:
      log("friendship established\r\n");

      //Start the sensor data timer when friendship is established.
      gecko_cmd_hardware_set_soft_timer(TIMER_MS_2_TICKS(10000),
                                        SENSOR_DATA_TIMER,
                                        0);
      break;

    case gecko_evt_mesh_lpn_friendship_failed_id:
      log("friendship failed\r\n");
      // try to establish friendship again in 2 seconds
      result = gecko_cmd_hardware_set_soft_timer(TIMER_MS_2_TICKS(3000),
                                                 FRIEND_FIND_TIMER,
                                                 SINGLE_SHOT)->result;
      if (result) {
        log("timer failure?!  0x%x\r\n", result);
      }
      break;

    case gecko_evt_mesh_lpn_friendship_terminated_id:
      log("friendship terminated\r\n");
      if (num_mesh_proxy_conn == 0) {
          // try to establish friendship again in 2 seconds
        result = gecko_cmd_hardware_set_soft_timer(TIMER_MS_2_TICKS(2000),
                                                   FRIEND_FIND_TIMER,
                                                   SINGLE_SHOT)->result;
        if (result) {
          log("timer failure?!  0x%x\r\n", result);
        }
      }
      break;

    default:
      break;
  }
}
```

Kuva 10. Ystävyyssuhteen muodostamista ohjaava switch-case-rakenne lpn.c-tiedostossa. Anturidatan ajastin käynnistetään tässä funktiossa.

Datan lähetys tapahtuu sensor.c-tiedostossa, jossa konfiguroidaan laitteen lähettämä data mesh-standardissa määritellyiksi Ambient temperature- ja Input voltage-datatyypeiksi. Kun 10 sekunnin pituinen ajastin on kulunut loppuun, ohjelma siirtyy switch-case rakenteen kautta sensor.c-tiedostossa sijaitsevaan julkaisufunktioon (**Kuva 11.**), joka kerää anturilta lämpötiladatan, laskee tulojännitteen ja julkaisee ne Friend-laitteille. Kehitysalustan lämpötila-anturia ei tarvinnut ohjelmoida erikseen, sillä anturille löytyi valmiit ajurit kehitysalustan mallikoodista. Kuvan 12 vuokaavio havainnollistaa mittauslaitteen ohjelman kulkua.

```
void handle_sensor_server_publish_event(
  struct gecko_msg_mesh_sensor_server_publish_evt_t *pEvt)
{
  log("evt:gecko_evt_mesh_sensor_server_publish_id\r\n");
  uint8_t sensor_data[15];
  uint8_t len = 0;

  temperature_8_t temperature = get_temp();
  len += mesh_sensor_data_to_buf(PRESENT_AMBIENT_TEMPERATURE,
                                 &sensor_data[len],
                                 (uint8_t*)&temperature);

  voltage_t present_input_voltage_t = get_battery_level();
  len += mesh_sensor_data_to_buf(PRESENT_INPUT_VOLTAGE,
                                 &sensor_data[len],
                                 (uint8_t*)&present_input_voltage_t);

  if (len > 0) {
    gecko_cmd_mesh_sensor_server_send_status(SENSOR_ELEMENT,
                                             PUBLISH_ADDRESS,
                                             IGNORED,
                                             NO_FLAGS,
                                             len, sensor_data);
  }
}
```

Kuva 11. Funktio, joka julkaisee lämpötilan ja tulojännitteen



Kuva 12. Mittauslaitteen ohjelman vuokaavio

Laitteen virrankulutusta voidaan seurata WSTK-mainboard-kehitysalustassa olevan AEM-ominaisuuden avulla. Thunderboard EFR32BG22 kytketään WSTK-mainboardiin käyttäen tarkoitukseen suunniteltua Mini Simplicity Connector-johtoa. Matalan virrankulutuksen mahdollistamiseksi ohjelmassa on myös konfiguroitava kello, jota laite pystyy käyttämään unitilassa. Kehitysalustassa käytetään matalataajuista kristallioskillaattoria mahdollistamaan unitila.

Kuvasta 13 nähdään, että AEM:n mittaama virrankulutus mittauslaitteelle on 35 mikroampeeria. Kuvassa näkyvän suurimman virtapiikin kohdalla laite herää ja julkaisee anturidatan Friend-laitteille. Pariston kesto voidaan laskea kaavan 1 avulla. Jos prototyyppilaite saataisiin toimimaan tällä samalla virrankulutuksella, se kestäisi 220mAh-kapasiteettisella CRC2032-paristolla noin 6285 tuntia (261 vuorokautta) ideaalisissa olosuhteissa.

$$Akun\ kesto\ tunteina\ (h) = \frac{akun\ kapasiteetti(mAh)}{kuorman\ virta(mA)} \quad (1)$$

Kuva 13. Mittauslaitteen virrankulutus Advanced Energy Monitor-näkymässä

5.2.2 Friend-laitteen ohjelmointi

Friend-laitteen ohjelman rakennetaan sensor client-malliohjelman pohjalle. Ohjelman alkuperäinen toiminto on tulostaa toisen laitteen lähettämää dataa näytölle ja konsoliin. Kehitysalustan nappia painamalla voidaan vaihtaa tulostetun datan tyyppiä. WSTK mainboard tukee tätä malliohjelmaa, mutta halutun toiminnallisuuden saavuttamiseksi siihen on lisättävä friend-ominaisuus ja laitteen tulostama data pitää rajata lämpötilaan ja tulojännitteeseen. Bluetooth-kommunikaation tapahtumat

hoidetaan tässäkin ohjelmassa odottamalla tapahtumia pääsilmukassa, josta siirrytään switch-case-rakenteeseen.

Ohjelmaan lisätään Friend-ominaisuuden alustusfunktio ja switch-case-rakenteeseen haarat, jotka tulostavat konsoliin tiedon ystävyyssuhteen muodostamisesta tai päättymisestä (**Kuva 14.**). Dataa vastaanottaessa ohjelma siirtyy kuvan 15 mukaiseen haaraan, josta ohjelma kulkeutuu tulostusfunktioon.

```
case gecko_evt_mesh_friend_friendship_established_id:
  log("evt gecko_evt_mesh_friend_friendship_established, lpn_address=%x\r\n", pEvt->data.evt_mesh_friend_friendship_established.lpn_address);
  DI_Print("FRIEND", DI_ROW_FRIEND);
  break;

case gecko_evt_mesh_friend_friendship_terminated_id:
  log("evt gecko_evt_mesh_friend_friendship_terminated, reason=%x\r\n", pEvt->data.evt_mesh_friend_friendship_terminated.reason);
  DI_Print("NO LPN", DI_ROW_FRIEND);
  break;
```

Kuva 14. Ystävyyssuhteen muodostuksesta ja päättymisestä ilmoittavat haarat

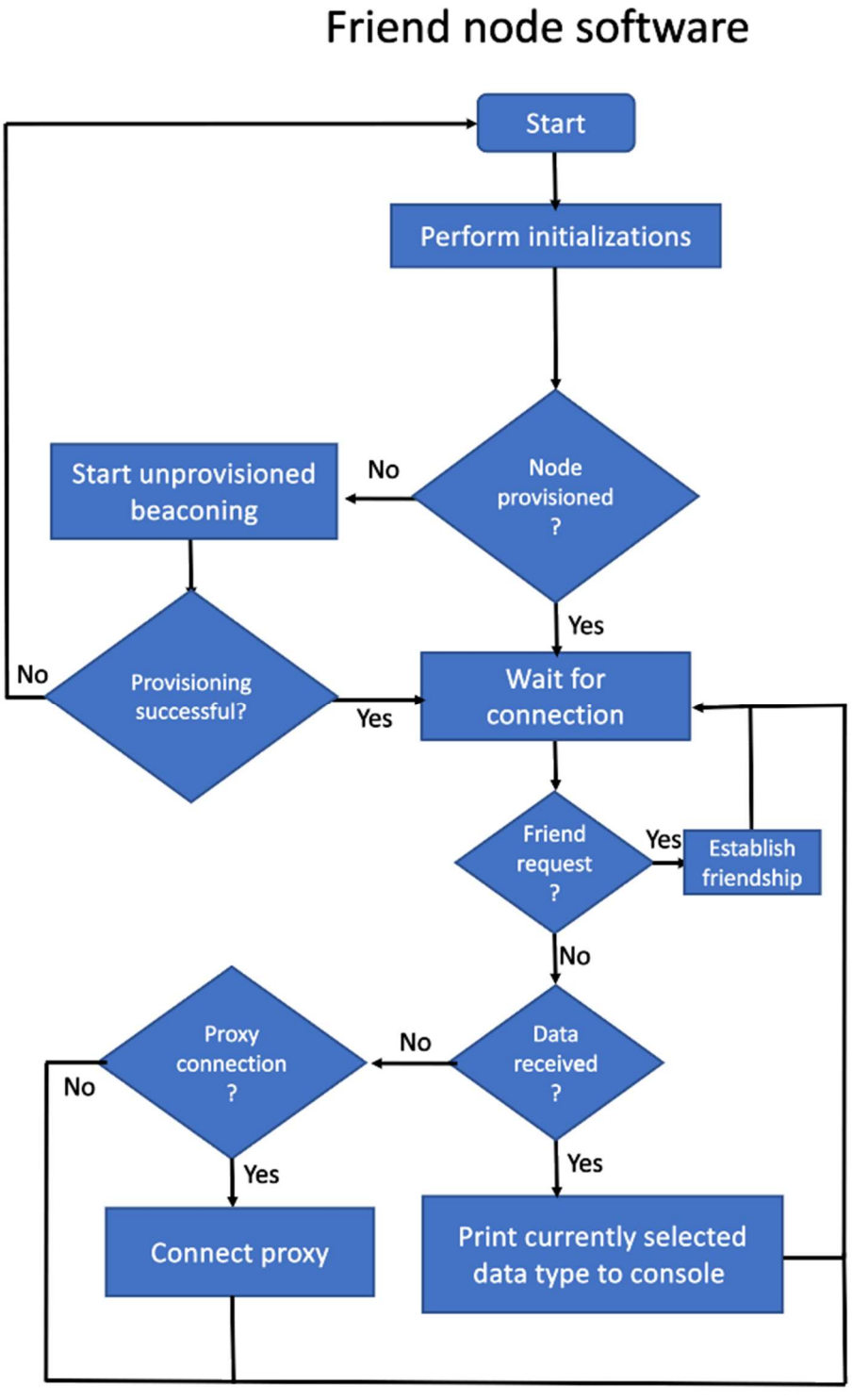
```
case gecko_evt_mesh_sensor_client_status_id:
    log("sensor data received\r\n");
    handle_sensor_client_events(pEvt);
    break;
```

Kuva 15. Haara datan vastaanottamiselle

Datan tulostus tapahtuu sensor_client.c -tiedostossa, josta poistetaan alkuperäisten antureiden datan tulostus ja lisätään lämpötilan ja tulojännitteen tulostus. Ohjelmassa säilytettiin toiminto, jonka avulla voidaan vaihtaa tulostettavaa arvoa lämpötilan ja tulojännitteen välillä painonappia painamalla. Datatyypin vaihto ja datan tulostus tapahtuvat kuvan 16 mukaisen switch-case-rakenteen avulla. Valmis ohjelma toimii kuvan 17 vuokaavion mukaisesti.

```
switch (property_id) {
case PRESENT_INPUT_VOLTAGE:
    if (property_len == 2) {
        mesh_device_property_t property =
                mesh_sensor_data_from_buf(
                        PRESENT_INPUT_VOLTAGE,
                        property_data);
        voltage_t present_input_voltage_t =
                property.voltage;

        if (present_input_voltage_t
                == (voltage_t) 0xFFFF) {
            snprintf(tmp, 21, "Adr %4f Voltage   N/K",
                    address_table[sensor]);
        } else {
            snprintf(tmp, 21, "Voltage %d",
                    present_input_voltage_t);
            log("Adr %4x Voltage %d \r\n",
                    address_table[sensor],
                    present_input_voltage_t);
        }

    } else {
        snprintf(tmp, 21, "Adr %f Voltage   N/A",
                address_table[sensor]);
    }
    DI_Print(tmp, DI_ROW_SENSOR_DATA + sensor);
    break;

case PRESENT_AMBIENT_TEMPERATURE:
    if (property_len == 1) {
        mesh_device_property_t property =
                mesh_sensor_data_from_buf(
                        PRESENT_AMBIENT_TEMPERATURE,
                        property_data);
        temperature_8_t temperature =
                property.temperature_8;

        if (temperature == (temperature_8_t) 0xFF) {
            snprintf(tmp, 21, "Adr %4x Temp    N/K",
                    address_table[sensor]);
        } else {
            snprintf(tmp, 21, "Adr %4x Temp %3d.%1dC",
                    address_table[sensor],
```

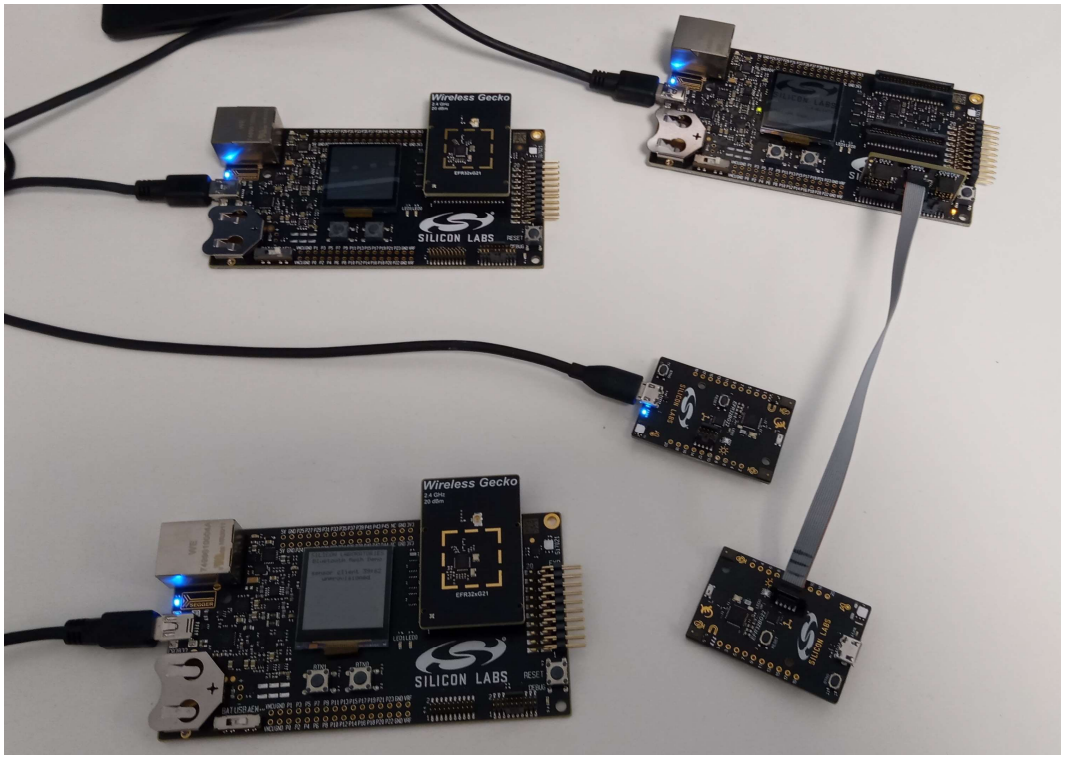
Kuva 16. Datan tulostamisesta vastaava switch-case-rakenne.



Kuva 17. Friend-laitteen ohjelman vuokaavio

5.3 Mallijärjestelmän toiminta

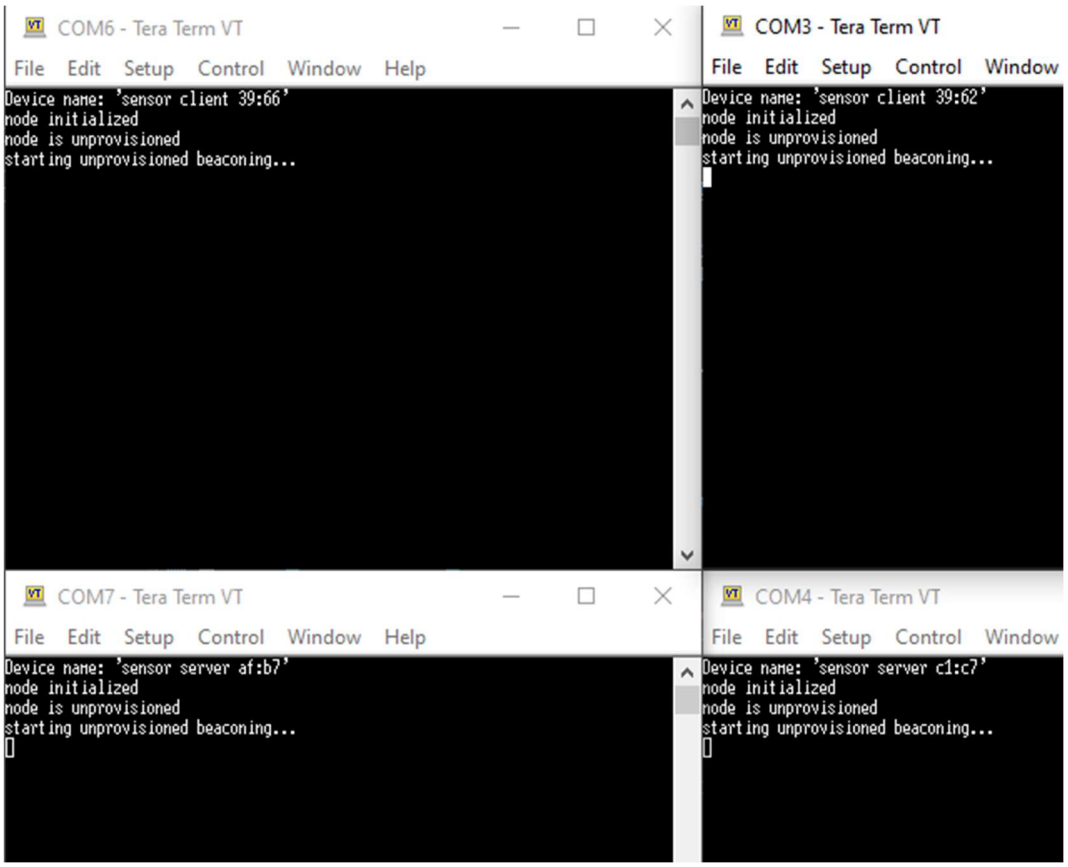
Laitteet lähettävät tapahtumista debug-viestejä, joita voidaan seurata konsoliyhteyden avulla Tera Term-ohjelmalla. Toimintaa havainnollistavissa kuvissa kaksi alinta Tera Term-ikkunaa kuuluvat mittauslaitteille ja kaksi ylintä Friend-laitteille. Kaikki laitteet saavat virran USB-johdon kautta tietokoneesta. Kuvassa 18 kaikki laitteet ovat ohjelmoituna ja valmiina provisiointiin. Yksi mittauslaite on kytketty WSTK mainboard-kehitysalustaan virranmittausta varten.



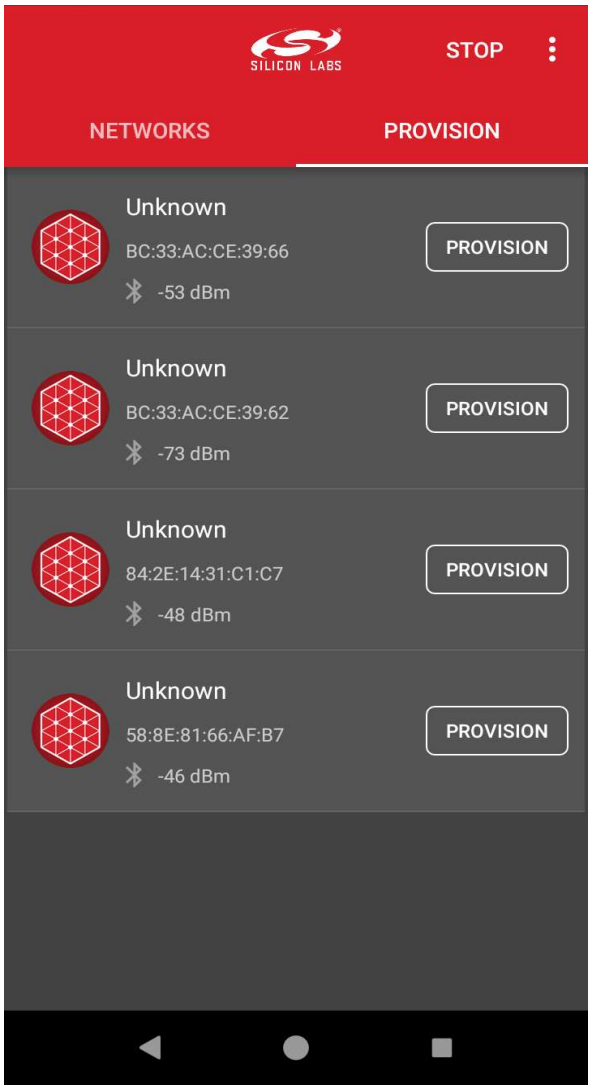
Kuva 18. Järjestelmän laitteet

5.3.1 Provisiointi

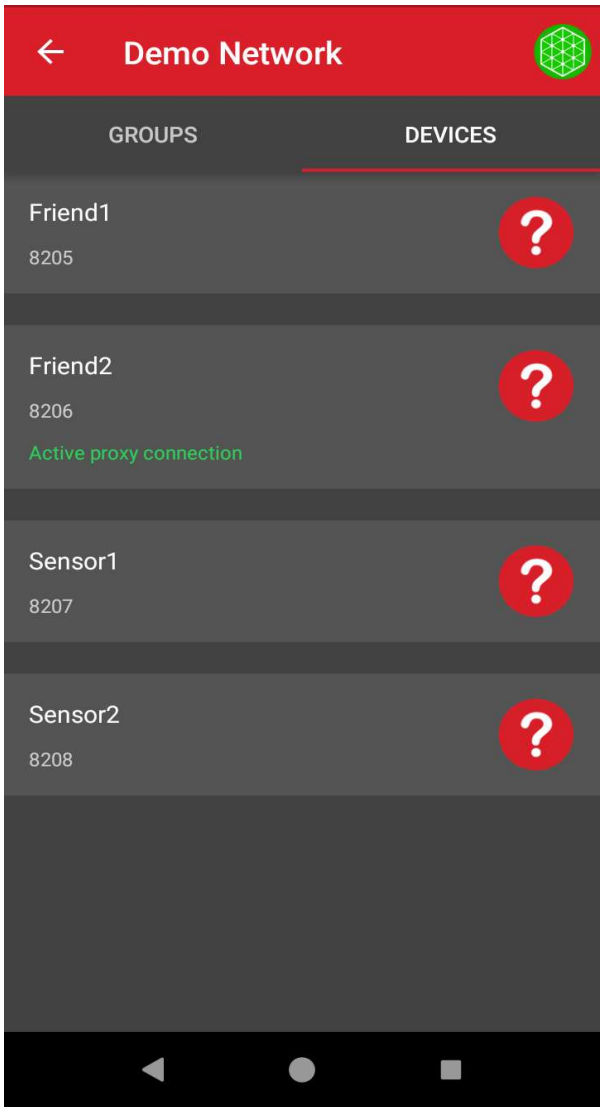
Kun kaikki laitteet ovat ohjelmoitu, järjestelmä voidaan provisioida. Provisioimattomat laitteet lähettävät unprovisioned beacon-paketteja (**Kuva 19.**), joiden avulla puhelinsovellus tunnistaa laitteet. Provisioinnin yhteydessä laitteet lisätään Demo Network-verkon alla olevaan Demo Group-ryhmään ja niille annetaan nimet. Kuvassa 20 provisioimattomat laitteet näkyvät sovelluksen provisiointinäkymässä. Provisioinnin ja nimeämisen jälkeen laitteet näkyvät sovelluksessa Demo Network-nimisen verkon alla kuvan 21 mukaisesti.



Kuva 19. Provisioimattomien laitteiden debug-viestit Tera Term-konsolissa



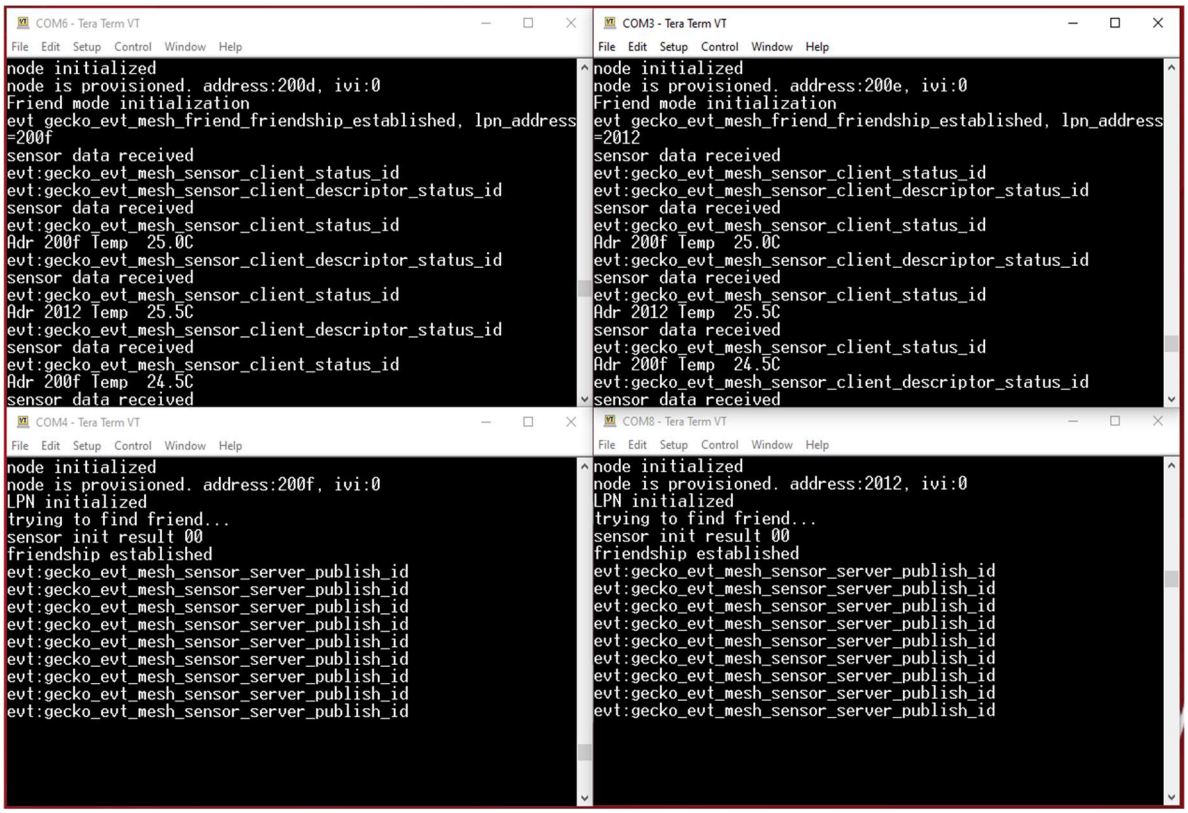
Kuva 20. Provisioimattomat laitteet puhelinsovelluksessa



Kuva 21. Laitteet provisioituna Demo Network-nimiseen mesh-verkkoon. Yhteys verkkoon tapahtuu proxy-yhteydellä tässä tapauksessa Friend2-laitteen kautta.

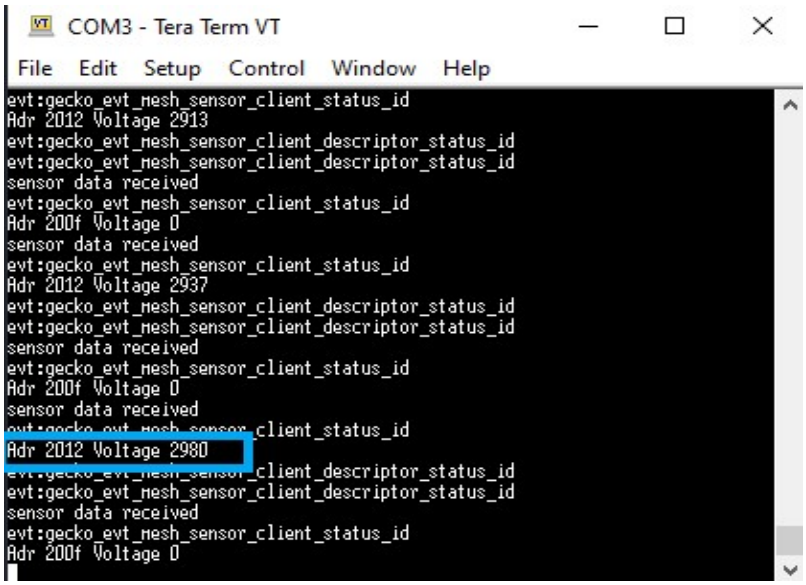
5.3.2 Laitteiden kommunikointi

Laitteiden provisioinnin jälkeen järjestelmä aloittaa toimintansa. Mittauslaitteet muodostavat onnistuneesti ystävyyssuhteen lähimmän Friend-laitteen kanssa ja aloittavat datan lähetyksen. Friend-laitteet vastaanottavat datan mittauslaitteilta tulostavat sen konsoliin. Kuvassa 22 nähdään laitteiden kommunikaatiota Tera Term-ohjelman näkymästä. Kehitysalustojen mittaama lämpötila on noin kaksi astetta korkeampi kuin ympäristön oikea lämpötila, sillä USB-johdolla toimiva debug-yhteys luo kehitysalustoille hieman lämpöä.



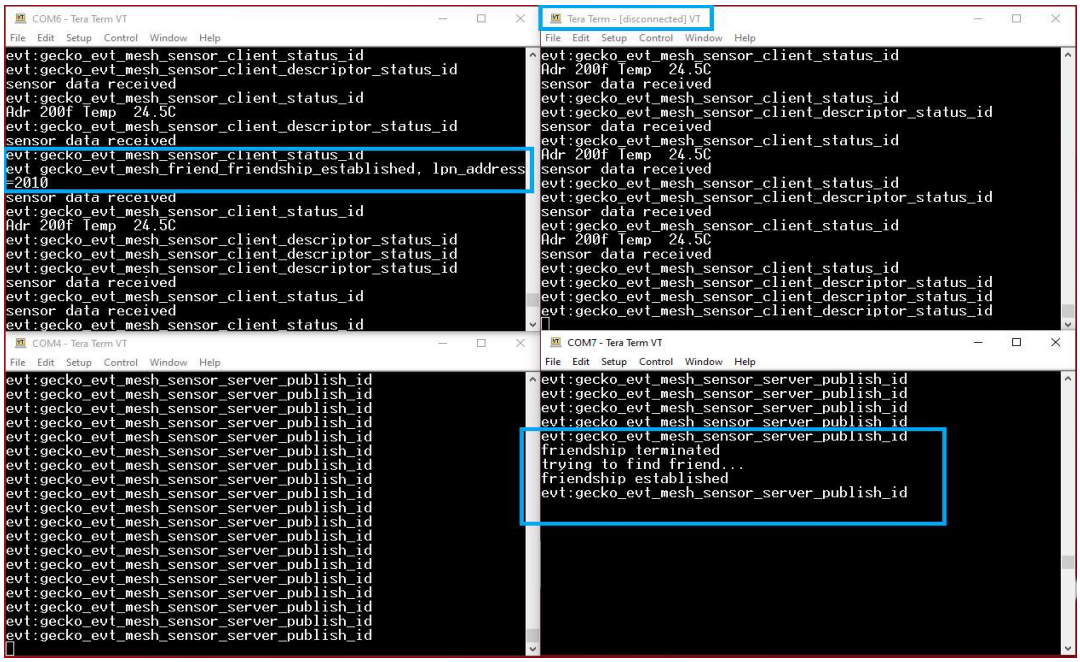
Kuva 22. Friendship-tapahtuma ja datan lähetys laitteiden välillä

Painamalla Friend-laitteen painonappia, sen tulostama data voidaan muuttaa lämpötiladatasta tulojännitteeksi. Kuvassa 23 toiseen mittauslaitteeseen on kytketty virtalähde kuvaamaan paristoa ja konsolista voidaan lukea sille syötetty jännite millivoltteina.



Kuva 23. Tulojännitteen tason tulostus

Jos toisesta Friend-laitteesta otetaan virrat pois, mittauslaitteen on muodostettava uusi ystävyyssuhde toiminnan jatkumiseksi. Kuvassa 24 toinen Friend-laite pudotetaan verkosta ja mittauslaite muodostaa yhteyden automaattisesti jäljellä olevaan Friend-laitteeseen.



Kuva 24. Uuden ystävyyssuhteen muodostus toisen Friend-laitteen sammuessa.

PTI-ominaisuuden avulla voidaan seurata ystävyyssuhteen muodostamisen aikana liikkuvia Bluetooth-paketteja. Kuvassa 25 on PTI-näkymä, jossa LPN-laite lähettää ystävyyspyynnön Friend-laitteelle. Pyynnön jälkeen laitteet muodostavat ystävyyssuhteen ja Friend-laite lisää LPN-laitteen tilauslistaansa. Yläpuolella näkyvät tapahtumat ovat korkean tason kuvaus silmukoiden kommunikaatiosta ja alapuolella näkyvät yksittäiset paketit.

Time	Duration	Summary	NWK Src	NWK Dest	P#
23.392177	5.295	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	2011	9
23.427100	5.413	BT Mesh Control Message: Friend Update	BC 33 AC CE 39 62	200D	3
23.429718	0.026	BT Mesh Control Message: Friend Subscription List Add	84 2E 14 31 C1 C7	2011	6
23.466275	0.002	BT Mesh Control Message: Friend Subscription List Confirm	BC 33 AC CE 39 62	200D	1
27.749898	1.077	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	2011	9
28.676058	0.034	BT Mesh Packet: Access Message	84 2E 14 31 C1 C7	C000	18
28.676058	0.034	BT Mesh Packet: Access Message	84 2E 14 31 C1 C7	C000	18
37.615700	0.002	BT Mesh Control Message: Friend Offer	BC 33 AC CE 39 62	200D	1
37.620788	5.294	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	2011	9
37.654693	5.413	BT Mesh Control Message: Friend Update	BC 33 AC CE 39 62	200D	3
37.657332	0.003	BT Mesh Control Message: Friend Subscription List Add	84 2E 14 31 C1 C7	2011	3
37.671139	0.002	BT Mesh Control Message: Friend Subscription List Confirm	BC 33 AC CE 39 62	200D	1
41.954723	1.099	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	2011	9
42.903623	0.034	BT Mesh Packet: Access Message	84 2E 14 31 C1 C7	C000	18

total:410 shown:391 Decoders: Bluetooth Low Energy, Default Profile

Time	Type	Summary	MAC Src	MAC Dest
34.423465	Packet	BLE LL - Adv Non-connectable Indication	11 3A 81 A6 89 3D	
34.435382	Packet	BLE LL - Adv Non-connectable Indication	06 D6 4E 21 14 1D	
34.460864	Packet	BLE LL - Adv Indication	64 1C AE 49 46 CF	
34.469069	Packet	BLE LL - Adv Non-connectable Indication	25 4E A7 76 AC 9E	
34.474110	Packet	BLE LL - Adv Non-connectable Indication	1C 65 A2 69 8F 02	
34.483366	Packet	BLE LL - Adv Indication	64 1C AE 49 46 CF	
37.510230	Packet	BT Mesh Control Message: Friend Request	84 2E 14 31 C1 C7	
37.510791	Packet	BT Mesh Control Message: Friend Request	84 2E 14 31 C1 C7	
37.511353	Packet	BT Mesh Control Message: Friend Request	84 2E 14 31 C1 C7	
37.615700	Packet	BT Mesh Control Message: Friend Offer	BC 33 AC CE 39 62	
37.616557	Packet	BLE LL - Adv Non-connectable Indication	14 FD D0 71 95 2B	
37.620788	Packet	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	
37.621278	Packet	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	
37.621769	Packet	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	
37.643527	Packet	BT Mesh Control Message: Friend Poll	84 2E 14 31 C1 C7	

Kuva 25. Friend request-tapahtuma Packet Trace Interface-näkymässä.

5.4 Mittauslaitteen ohjelman siirtäminen prototyyppilevylle

Jotta ohjelma saadaan toimimaan prototyyppilevyllä, mikro-ohjaimen nastat ovat määriteltävä oikein ja uuden lämpötila-anturin toiminta on ohjelmoitava. Prototyyppilevyllä on sama mikro-ohjain kuin kehitysalustassa, joten ohjelman siirtäminen ei vaadi suuria muutoksia.

Lämpötila-anturille luodaan c- ja header-tiedostot, joihin sen toiminnallisuus ohjelmoidaan datalehteä apuna käyttäen /13/. Anturi on kytketty kiinni mikro-ohjaimen I2C-tiedonsiirtoväylään. I2C on kaksisuuntainen tiedonsiirtoväylä, joka toimii data- ja kellolinjojen avulla. Väylä toimii ohjaaja-oheislaite-periaatteella, jossa tässä tapauksessa anturi on oheislaite ja mikro-ohjain ohjaaja /19/.

MLX90614-anturilta halutaan lähettää samanaikaisesti ympäristön lämpötilatieto sekä infrapunalla mitattu kappaleen lämpötilatieto lattian lämmön mittaamista varten. Tämän mahdollistamiseksi julkaisufunktiota on hieman muutettava. Kuvassa 11 näkyvä get_temp()-funktiokutsu korvataan kutsuilla uudelle anturille luotuihin get_temp_object()- ja get_temp_ambient()-funktioihin (**Kuva 26.**). Lämpötila-anturin koodi on nähtävissä liitteissä 3 ja 4.

Mesh-kirjastoa käytettäessä data syötetään datapuskuriin, jonka kautta se julkaistaan. Tämä puskuri tukee kuitenkin ainoastaan 8-bittisiä muuttujia. joilla 0,1 celsiusasteen resoluutio ei ole aina mahdollista. Tämän vuoksi lämpötilatiedot ovat jaettava kahteen 8-bittiseen osaan, jotka yhdistetään Friend-laitteen päässä takaisin kokonaisiksi lämpötila-arvoiksi (**Kuva 27.**). Datapuskuri täytettiin aluksi kuvassa 11 näkyvän mesh_sensor_data_to_buf-funktion avulla, mutta tämä funktio ei tue kahden eri arvon lisäämistä saman datatyypin alle. Ratkaisu ongelmaan on täyttää datapuskuri manuaalisesti kuvan 26 mukaisesti.

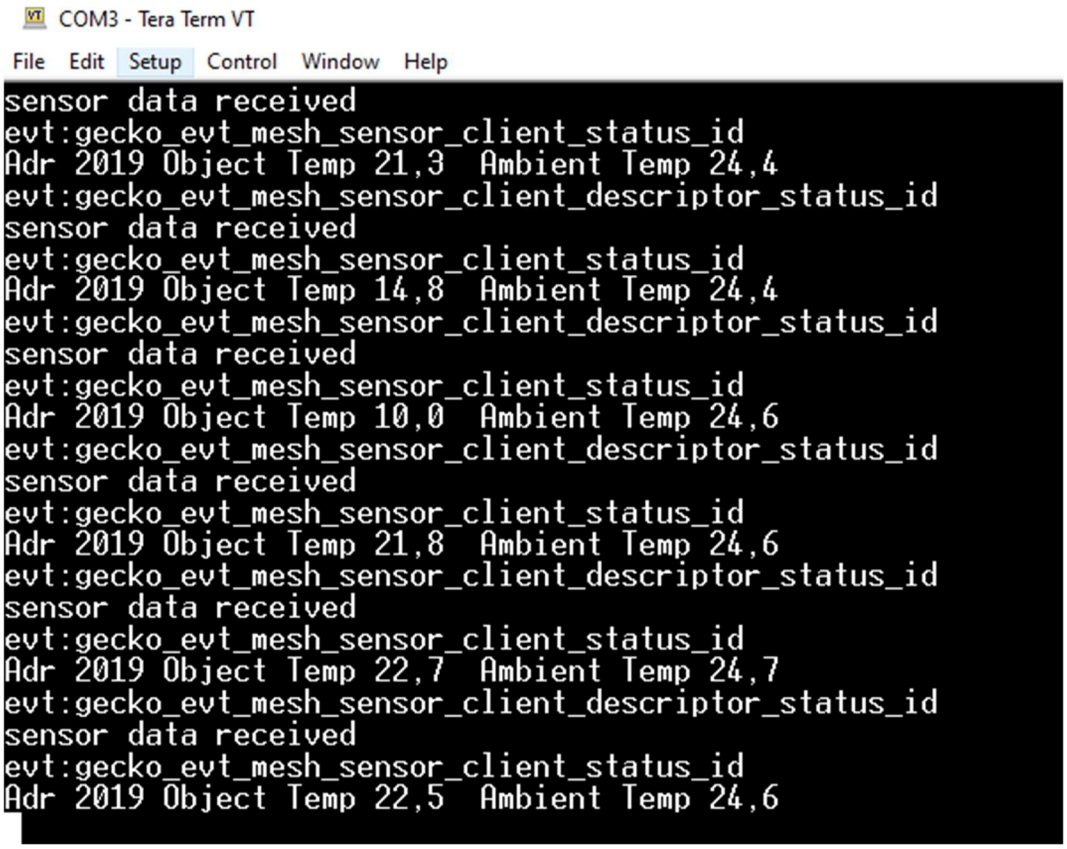
```
void handle_sensor_server_publish_event(
  struct gecko_msg_mesh_sensor_server_publish_evt_t *pEvt)
{
  log("evt:gecko_evt_mesh_sensor_server_publish_id\r\n");
  uint8_t sensor_data[7];
  uint8_t len = 7;

  temperature_t temperature_object = get_temp_object();
  temperature_t temperature_ambient = get_temp_ambient();

  //split 16-bit temperature values into bytes
  temperature_8_t obj1 = ((temperature_object >> 0) & 0xFF);
  temperature_8_t obj2 = ((temperature_object >> 8) & 0xFF);
  temperature_8_t amb1 = ((temperature_ambient >> 0) & 0xFF);
  temperature_8_t amb2 = ((temperature_ambient >> 8) & 0xFF);

  sensor_data[0] = PRESENT_AMBIENT_TEMPERATURE;
  sensor_data[1] = 0;
  sensor_data[2] = 1;               //first 3 variables in sensor_data
  sensor_data[3] = obj1;
  sensor_data[4] = obj2;
  sensor_data[5] = amb1;
  sensor_data[6] = amb2;

  if (len > 0) {
    gecko_cmd_mesh_sensor_server_send_status(SENSOR_ELEMENT,
                                             PUBLISH_ADDRESS,
                                             IGNORED,
                                             NO_FLAGS,
                                             len, sensor_data);
  }
}
```

Kuva 26. Uusi julkaisufunktio

```
case PRESENT_AMBIENT_TEMPERATURE:
    if (property_len == 1) {

        //Concatenate temperature values
        temperature_t temperature_object = (*property_data | *property_data2 << 8);
        temperature_t temperature_ambient = (*property_data3 | *property_data4 << 8);
        double obj = (double)temperature_object / 10;
        double amb = (double)temperature_ambient / 10;

        //Logging does not support float values, so split them into integers
        temperature_8_t obj1 = obj * 10 / 10;
        temperature_8_t obj2 = obj * 10 - obj1 * 10;
        temperature_8_t amb1 = amb * 10 / 10;
        temperature_8_t amb2 = amb * 10 - amb1 * 10;


        if ((temperature_object == (temperature_8_t) 0xFF) && (temperature_ambient == (temperature_8_t) 0xFF)) {
            snprintf(tmp, 21, "Adr %4x Temp    N/K",
                    address_table[sensor]);
        } else {
            snprintf(tmp, 39, "Adr %4x Obj %d,%d Amb %d,%d ",   //
                    address_table[sensor],
                    obj1, obj2, amb1, amb2);

            log("Adr %4x Object Temp %d,%d  Ambient Temp %d,%d \r\n",
                    address_table[sensor],
                    obj1, obj2, amb1, amb2);
        }
    } else {
        snprintf(tmp, 21, "Adr %4x Temp    N/A",
                address_table[sensor]);
    }
```

Kuva 27. Purettujen lämpötila-arvojen yhdistäminen Friend-laitteen ohjelmassa

Ohjelma siirretään levylle WSTK mainboard-kehitysalustan kautta käyttäen Mini Simplicity Connector-johtoa, jonka jälkeen laite voidaan provisioida Bluetooth mesh-sovelluksella verkkoon. Provisioinnin jälkeen laite toimii samalla tavalla kuin kehitysalusta, eli se muodostaa ystävyyssuhteen Friend-laitteen kanssa ja alkaa lähettämään lämpötila-arvoa 10 sekunnin välein. Kuvassa 28 laitteen anturin mittaamaa kappaletta jäähdytettiin kylmäsprayn avulla ja kappaleen lämpötilan muutos voidaan havaita Friend-laitteen Tera Term-näkymästä. Object Temp-arvo on kappaleen lämpötilatieto ja Ambient Temp ympäristön lämpötilatieto. Lämpötiladata tulostetaan celsiusasteina.



Kuva 28. Friend-laitteen Tera Term-näkymässä.

6 KEHITTÄMISTARPEET JA JATKOKEHITYS

Mallijärjestelmän suunnittelun ja kehittämisen aikana on ilmennyt ideoita järjestelmän jatkokehitykseen. Nämä asiat tulee ottaa huomioon lopullisen järjestelmän kehityksessä.

Mesh-verkon provisointiin on kehitetty uusi self provisioning-ratkaisu. Tällä tekniikalla laitteet pystyvät provisioimaan itsensä, kunhan niille syötetään tarvittavat salausavaimet. Muut tiedot provisiointia varten pystytään hoitamaan laitteen ohjelmassa. Tämän ratkaisun avulla provisiointia varten ei välttämättä tarvitse kehittää erillistä puhelinsovellusta, jos salausavaimet pystytään siirtämään laitteille muulla tavalla /17/.

Lopullisessa järjestelmässä on myös tutkittava tarkemmin proxy-ominaisuutta, sillä huoneiston lämpötilaa on pystyttävä hallitsemaan puhelimella. Lämpötilan ohjauslaitteeseen on siis saatava proxy-protokollan avulla yhteys puhelinsovelluksesta, mutta puhelimen ei tarvitse olla osana mesh-verkkoa. Tässä sovelluksessa riittää, että puhelinsovelluksesta voidaan siirtää ohjauslaitteelle halutun lämpötilan arvo ja lukea tämänhetkinen lämpötila.

7 JOHTOPÄÄTÖKSET

Opinnäytetyön lopputuloksena on toimiva malli termostaattijärjestelmästä. Mallin ohjelmistoa kehitettäessä pidettiin mielessä sen siirrettävyys, jotta sitä voidaan hyödyntää jatkossa lopullisessa tuotteessa. Bluetooth mesh-teknologian teoria ja toimintaperiaate selkeni myös paljon projektin aikana. Mallijärjestelmästä tulee olemaan hyötyä tulevien mesh-laitteiden kehityksessä ja jatkokehitysideoita ilmeni työn aikana.

Prototyyppilevyn implementointi tutustutti minut sulautettujen järjestelmien kehitysprosessiin, jossa sovellusta hahmotellaan ensin kehitysalustalla. Ohjelman siirtämisessä piti ottaa huomioon erilaisia asioita, joista sain lisää näkemystä mikro-ohjainten ja oheislaitteiden ohjelmointiin.

Mallijärjestelmän suunnittelu ja kehittäminen soveltui mielestäni hyvin opinnäytetyöksi. Käytin paljon aikaa Bluetooth- ja Bluetooth mesh-teknologioiden opiskeluun, joten opin paljon uutta näistä aiheista. Opin paljon myös yleisestä sulautettujen järjestelmien kehityksestä. Tiedon löytäminen oli välillä hieman hankalaa, sillä Bluetooth mesh on melko uusi teknologia. Valmistajan dokumentointi ja Bluetooth mesh-standardi olivat lopulta kuitenkin riittävän kattavat mallijärjestelmän toteuttamiseen.

LÄHTEET

/1/ Mesh Profile Specification v1.0.1. Viitattu 8.3.2021.
Ladattavissa: https://www.bluetooth.com/specifications/specs/

/2/ Mesh Technology Overview. Viitattu 15.3.2021.
https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf

/3/ Bluetooth Mesh Glossary of Terms. Viitattu 17.3.2021.
https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/mesh-glossary/

/4/ Fundamental Concepts of Bluetooth Mesh Networking Part 2.
Viitattu 15.3.2021.
https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-2/

/5/ Designing with Bluetooth Mesh: Nodes and feature types. Viitattu 18.3.2021.
https://www.embedded.com/designing-with-bluetooth-mesh-nodes-and-feature-types/

/6/ Silicon Labs UG366 Mesh Node Configuration User's Guide.
Viitattu 22.3.2021.
https://www.silabs.com/documents/public/user-guides/ug366-bluetooth-mesh-node-configuration-users-guide.pdf

/7/ Provisioning a Bluetooth mesh network Part 1. Viitattu 15.3.2021.
https://www.bluetooth.com/blog/provisioning-a-bluetooth-mesh-network-part-1/

/8/ Silicon Labs Sensor Model Demo. Viitattu 14.4.2021.
https://www.silabs.com/documents/public/application-notes/an1186-understanding-bluetooth-mesh-sensor-model-demo.pdf

/9/ Silicon Labs UG151 EFR32MG1 User's Guide. Viitattu 16.4.2021.
https://www.silabs.com/documents/public/user-guides/ug151-brd4151a-user-guide.pdf

/10/ Silicon Labs UG415 Thunderboard EFR32BG22 User's Guide
Viitattu 26.4.2021
https://www.mouser.com/pdfDocs/ug415-sltb010a-user-guide.pdf

/11/ Silicon labs Mesh-ohjelmistokehityspaketin käyttöohje
https://www.silabs.com/documents/public/reference-manuals/bluetooth-le-and-mesh-software-api-reference-manual.pdf

/12/Silicon Labs Thunderboard EFR32BG22-kehitysalustan tiedot.
Viitattu 28.4.2021
https://www.silabs.com/development-tools/thunderboard/thunderboard-bg22-kit

/13/ Prototyyppilevyn lämpötila-anturin datalehti. Viitattu 29.4.2021
https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf

/14/ Bit Shifting. Luettu 20.4.2021
https://www.computerhope.com/jargon/b/bit-shift.htm#:~:text=A%20bit%20shift%20is%20a,the%20CPU%20than%20conventional%20math.

/15/ Bit Masks. Luettu 20.4.2021
https://www.arduino.cc/en/Tutorial/Foundations/BitMask

/16/ EFR32BG22-järjestelmäpiirin datalehti. Luettu 7.4.2021.
https://www.silabs.com/documents/public/user-guides/ug151-brd4151a-user-guide.pdf

/17/ Self-provisioining. Viitattu 26.4.2021.
https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2020/10/28/self_provisioningandconfigurationexample-J9sJ

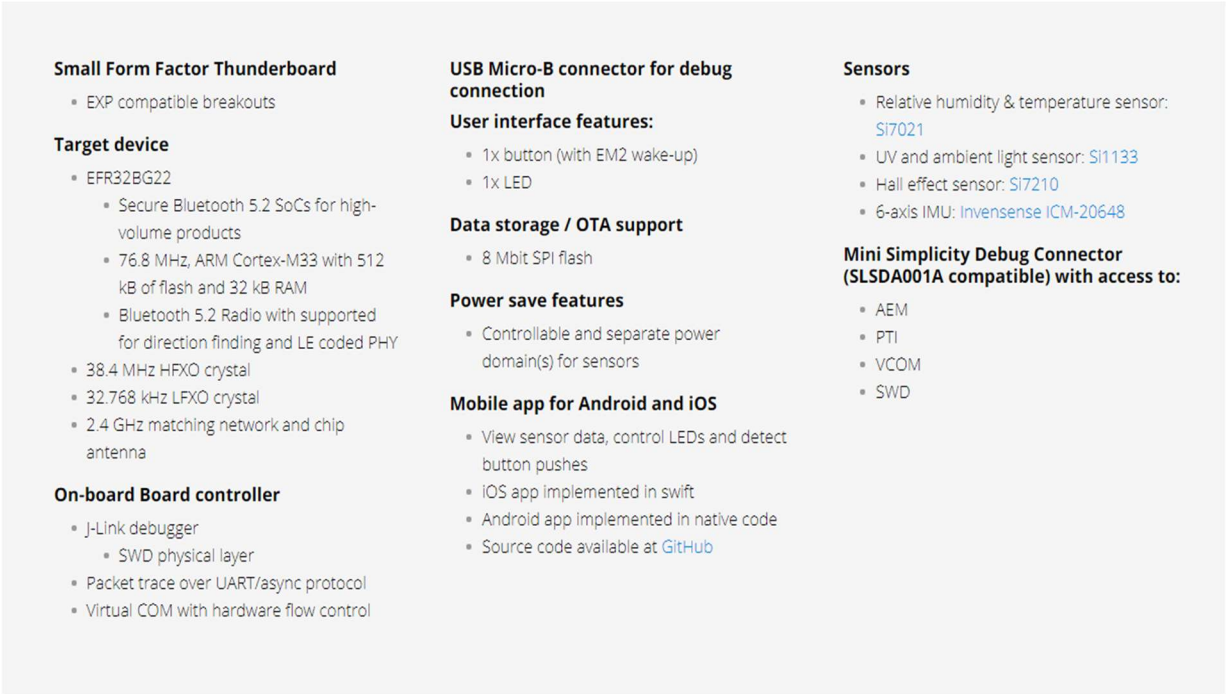
/18/ Mesh Networking: Friendship. Luettu 7.4.2021.
https://www.bluetooth.com/blog/bluetooth-mesh-networking-series-friendship/

/19/ I2C-protokolla. Viitattu 16.4.2021.
https://learn.sparkfun.com/tutorials/i2c/all

/20/ Mesh-puhelinsovellus. Luettu 1.3.2021.
https://www.silabs.com/developers/bluetooth-mesh-mobile-app

LIITTEET

LIITE 1



Small Form Factor Thunderboard

- EXP compatible breakouts

Target device

- EFR32BG22
  - Secure Bluetooth 5.2 SoCs for high-volume products
  - 76.8 MHz, ARM Cortex-M33 with 512 kB of flash and 32 kB RAM
  - Bluetooth 5.2 Radio with supported for direction finding and LE coded PHY
- 38.4 MHz HFXO crystal
- 32.768 kHz LFXO crystal
- 2.4 GHz matching network and chip antenna

On-board Board controller

- J-Link debugger
  - SWD physical layer
- Packet trace over UART/async protocol
- Virtual COM with hardware flow control

USB Micro-B connector for debug connection

User interface features:

- 1x button (with EM2 wake-up)
- 1x LED

Data storage / OTA support

- 8 Mbit SPI flash

Power save features

- Controllable and separate power domain(s) for sensors

Mobile app for Android and iOS

- View sensor data, control LEDs and detect button pushes
- iOS app implemented in swift
- Android app implemented in native code
- Source code available at GitHub

Sensors

- Relative humidity & temperature sensor: Si7021
- UV and ambient light sensor: Si1133
- Hall effect sensor: Si7210
- 6-axis IMU: Invensense ICM-20648

Mini Simplicity Debug Connector (SLSDA001A compatible) with access to:

- AEM
- PTI
- VCOM
- SWD

Kuva 29. Thunderboard EFR32BG22-kehitysalustan tiedot /12/

LIITE 2

BRD4180A RADIO BOARD FEATURES

- EFR32xG21 Wireless Gecko Wireless SoC with 1024 kB Flash and 96 kB RAM (EFR32MG21A020F1024IM32).
- Inverted-F PCB antenna (2.4 GHz band)
- 2x user color LEDs (red and green)

WIRELESS STK MAINBOARD FEATURES

- Advanced Energy Monitor
- Packet Trace Interface
- Virtual COM port
- SEGGER J-Link on-board debugger
- External device debugging
- Ethernet and USB connectivity
- Low power 128x128 pixel Memory LCD-TFT
- User LEDs / pushbuttons
- 20-pin 2.54 mm EXP header
- Breakout pads for Wireless SoC I/O
- CR2032 coin cell battery support

SOFTWARE SUPPORT

- Simplicity Studio™
- Energy Profiler
- Network Analyzer

Kuva 30. WSTK mainboard-kehitysalustan ja EFR32MG21-radiokortin tiedot /9/

LIITE 3

Prototyyppilevyn lämpötila-anturin c-tiedosto

```
//Driver for the MLX90614 infrared temperature sensor

#include "i2cspm.h"
#include "ir_sensor.h"
#include <stdio.h>

temperature_t get_temp_object(void)
{
    temperature_t object_temperature = 0;
    temperature_t temperature = VALUE_IS_NOT_KNOWN;

    measure_temp_ir(&object_temperature);
    temperature = object_temperature;

    return temperature;
}

temperature_t get_temp_ambient(void)
{
    temperature_t ambient_temperature = 0;
    temperature_t temperature = VALUE_IS_NOT_KNOWN;

    measure_temp_ambient(&ambient_temperature);
    temperature = ambient_temperature;

    return temperature;
}

void measure_temp_ir(temperature_t *temp)
{
    uint8_t returndata[2] = {0};
    float value;

    i2c_read(MLX90614_I2CADDR, MLX90614_TOBJ1, returndata, 2);

    value = (returndata[0] | returndata[1] << 8);

    value *= .02;
    value -= 273.15;
    value *= 10;

    *temp = value;
}
```

```
void measure_temp_ambient(temperature_t *temp)
{
    uint8_t returndata[2] = {0};
    float value;

    i2c_read(MLX90614_I2CADDR, MLX90614_TA, returndata, 2);

    value = (returndata[0] | returndata[1] << 8);

    value *= .02;
    value -= 273.15;
    value *= 10;

    *temp = value;
}

/*Read bytes from slave device starting at the target address*/
void i2c_read(uint16_t slaveAddress, uint8_t targetAddress,
uint8_t *rxBuff, uint8_t numBytes)
{
  // Transfer structure
  I2C_TransferSeq_TypeDef i2cTransfer;
  I2C_TransferReturn_TypeDef result;

  // Initializing I2C transfer
  i2cTransfer.addr          = slaveAddress;
  i2cTransfer.flags         = I2C_FLAG_WRITE_READ;
  // must write target address before reading
  i2cTransfer.buf[0].data   = &targetAddress;
  i2cTransfer.buf[0].len    = 1;
  i2cTransfer.buf[1].data   = rxBuff;
  i2cTransfer.buf[1].len    = numBytes;

  result = I2C_TransferInit(I2C1, &i2cTransfer);

  // Sending data
  while (result == i2cTransferInProgress)
  {
    result = I2C_Transfer(I2C1);
  }
}

/*Write bytes to the target register on slave device*/
void i2c_write(uint16_t slaveAddress, uint8_t targetAddress,
uint8_t *txBuff, uint8_t numBytes)
{
  // Transfer structure
  I2C_TransferSeq_TypeDef i2cTransfer;
  I2C_TransferReturn_TypeDef result;
  uint8_t txBuffer[2 + 1];

  txBuffer[0] = targetAddress;
  for(int i = 0; i < numBytes; i++)
  {
      txBuffer[i + 1] = txBuff[i];
  }

  // Initializing I2C transfer
```

```
  i2cTransfer.addr          = slaveAddress;
  i2cTransfer.flags         = I2C_FLAG_WRITE;
  i2cTransfer.buf[0].data   = txBuffer;
  i2cTransfer.buf[0].len    = numBytes + 1;
  i2cTransfer.buf[1].data   = NULL;
  i2cTransfer.buf[1].len    = 0;

  result = I2C_TransferInit(I2C1, &i2cTransfer);

  // Sending data
  while (result == i2cTransferInProgress)
  {
    result = I2C_Transfer(I2C1);
  }
}
```

LIITE 4

Prototyyppilevyn lämpötila-anturin header-tiedosto

```
//Driver for the MLX90614 infrared temperature sensor
#include "mesh_device_properties.h"

#define MLX90614_I2CADDR 0x00    //i2c slave address of mlx90614

//RAM
#define MLX90614_RAWIR1 0x04     //raw ir data channel 1
#define MLX90614_RAWIR2 0x05     //raw ir data channel 2
#define MLX90614_TA 0x06         //ambient temperature
#define MLX90614_TOBJ1 0x07      //object 1 temperature
#define MLX90614_TOBJ2 0x08      //object 2 temperature
//EEPROM
#define MLX90614_TOMIN 0x20      //object temperature min register
#define MLX90614_TOMAX 0x21      //object temperature max register
#define MLX90614_PWMCTRL 0x22    //pwm configuration register
#define MLX90614_TARANGE 0x23    //ambient temperature range regis-
ter
#define MLX90614_EMISS 0x24      //emissivity register
#define MLX90614_CONFIG 0x25     //config register
#define MLX90614_ADDR 0x0E       //slave address register
#define MLX90614_ID1 0x3C        //ID registers for 1, 2, 3 and 4
#define MLX90614_ID2 0x3D
#define MLX90614_ID3 0x3E
#define MLX90614_ID4 0x3F

#define VALUE_IS_NOT_KNOWN  (0xFF) ///< Temperature value is not
known

temperature_t get_temp_object(void);
temperature_t get_temp_ambient(void);
void measure_temp_ir(temperature_t *temp);
void measure_temp_ambient(temperature_t *temp);
void i2c_read(uint16_t slaveAddress, uint8_t targetAddress,
uint8_t *rxBuff, uint8_t numBytes);
void i2c_write(uint16_t slaveAddress, uint8_t targetAddress,
uint8_t *txBuff, uint8_t numBytes);
```