Opinnäytetyö (AMK)

Tietotekniikka

Sulautetut ohjelmistot

2017

Teemu Myllylä

RAJAPINTOJEN KEHITYS SIGFOX-JÄRJESTELMÄPIIRILLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Sulautetut ohjelmistot

2017 | 57

Teemu Myllylä

RAJAPINTOJEN KEHITYS SIGFOX-JÄRJESTELMÄPIIRILLE

Sigfox-verkko on yksi monesta esineiden internetin laitteille kehitetyistä verkoista. Verkko toimii lisensoimattomilla taajuuksilla ja kilpailee muiden samankaltaisten verkkojen, kuten LoRan ja Weightlessin, kanssa. Sigfox-verkkoa operoi Suomessa Connected Finland, jonka ylläpitämään verkkoon tässä projektissa kehitetty järjestelmäpiiri on tarkoitus yhdistää.

Työn tavoitteena on kehittää tarvittavat rajapinnat uudelle Sigfox-järjestelmäpiirille. Vaadittuja toiminnollisuuksia ovat UART-kommunikaatio sekä GPIO-porttien hallinta. RCP Softwaren tuoteprototyypin toiminnollisuus on tarkoitus toistaa uudella Sigfox-piirillä käyttäen kehitettyjä rajapintoja. Tavoitteena on myös tutustua testivetoiseeen kehitykseen, hyödyntää sitä rajapintojen kehityksessä sekä pohtia sen tuomia hyötyjä ja haittoja sulautettujen ohjelmistojen kehityksessä.

Opinnäytetyön kehitysprojektissa luotiin rajapinnat RCP Softwaren omaan suunnitteluun perustuvalle Sigfox-järjestelmäpiirille. Rajapinnat kehitettiin UARTille, GPIO:lle sekä UARTin kautta vastaanotettaville komennoille. Projektin työstämisen aikana tutustuttiin testivetoisen kehityksen perusteisiin. Testivetoinen kehitys on ohjelmistonkehitysmuoto, jossa testitapaukset luodaan ennen ohjelmakoodia. Testivetoinen kehitys edistää ohjelmamoduulien keskinäistä riippumattomuutta, tuo varmuuden ohjelmakokonaisuuden toiminnasta ja toimii selkeänä dokumentaationa. Testivetoista kehitystä sovellettiin projektityön komentorajapinnan kehityksessä.

Projektin tuotoksena syntyi kolme toisistaan riippumatonta rajapintamoduulia. UART-rajapinta toteuttaa UART-viestien lähettämisen ja vastaanottamisen. GPIO-rajapinnalla pystytään hallitsemaan GPIO-portteja digitaalisina tuloina ja lähtöinä sekä asettamaan porteille keskeytyksiä. Komentorajapinta tulkitsee sille syötetyt komennot. Komennot määriteltiin vastaamaan toimeksiantajayrityksen tarpeita. Rajapinnat toteutettiin C-kielellä.

Projektityön tavoitteet täyttyivät lähes täysin. Kehitystyön aikana kohdattiin muutamia merkittäviä ongelmia, jotka venyttivät aikataulua. Ongelmat onnistuttiin selvittämään, mutta aikataulun venyttyä muutama ominaisuus jäi ajan puutteen vuoksi kehittämättä. Testivetoisen kehityksen todettiin olevan tehokas tapa ohjelmiston kehityksessä. Testivetoista kehitystä on mahdollista hyödyntää sulautettujen ohjelmistojen kehityksessä, kun tiedetään kohdelaitteiston tuomat rajoitukset.

ASIASANAT:

Sigfox, testivetoinen kehitys, esineiden internet, sulautetut ohjelmistot

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Software

2017 | 57

Teemu Myllylä

DEVELOPMENT OF INTERFACES FOR A SIGFOX MODULE

The Sigfox network is one of the many networks developed for IoT devices. The network uses unlicensed frequencies and competes with similar technologies, such as LoRa and Weightless. The Sigfox network in Finland is operated by Connected Finland.

The goal of the thesis is to develop the necessary interfaces for a new Sigfox module. The required functionalities include UART communication and GPIO management. The goal is also to get acquainted with test-driven development, to utilize it in the development of the interfaces, and to discuss its benefits and disadvantages in embedded software development.

In this thesis, interfaces were developed for a Sigfox module based on RCP Software design. Interfaces for UART communication, GPIO functionality and commands received and sent via UART were developed. During the development work for this thesis, the basics of test driven development were studied. Test driven development is a software development method where test cases are created before the code is written. Test driven development boosts software decoupling, developer confidence and works as clear documentation for the project. Test driven development was applied to the development of the command interface.

As a result, three decoupled interface modules were created. The UART interface implements the sending and receiving of UART messages. With the GPIO interface GPIO ports can be used as digital inputs or outputs and interrupts can be assigned for the ports. The command interface interprets given commands. Commands were defined to correspond the needs of the client company. The interfaces were implemented in the C language.

The goals for the project were achieved almost completely. During the development, a few major problems were encountered which delayed the schedule. The problems were solved but some functionalities were not implemented due to lack of time. Test driven development was found to be an effective way of developing software. Test driven development can be utilized in the development of embedded software when the restrictions of target hardware are known.

KEYWORDS:

Sigfox, IoT, test driven development, embedded software

SISÄLTÖ

KAAVAT

KUVAT

KUVIOT

TAULUKOT

KÄYTETYT LYHENTEET

ADC	Analog Digital Converter. Analogisen signaalin muunnos digitaaliseksi.
API	Application Programming Interface. Ohjelmointirajapinta, jonka kautta ohjelmat toimivat keskenään.
AXSDB	Virheenjäljittäjäsovellus On Semiconductorin järjestelmäpiireille.
DAC	Digital Analog Converter. Digitaalisen signaalin muunnos analogiseksi.
DebugLink	ON Semiconductorin kehittämien järjestelmäpiirien ja virheenjäljittäpiirin kommunikointiin käytetty viestintäprotokolla.
FIFO	First In First Out. Rekisterityyppi, jossa arvot poistuvat samassa järjestyksessä kuin saapuvat.
FROSC	Fast RC Oscillator. AX8052:n nopea 20 MHz:n RC-oskillaattori (ON Semiconductor 2016d, 36).
GPIO	General Purpose Input Output. Ohjelmoitava portti järjestelmäpiirillä.
HTTPS	Hypertext Transfer Protocol Secure. Salattu tiedonsiirtoprotokolla.
IRAM	Internal Random Access Memory. Prosessorin sisäinen RAM-muisti.
LibMF	Ohjelmointia helpottava kirjastokokoelma AX8052-mikro-ohjaimille (ON Semiconductor 2016c, 1).
LPWAN	Low Power Wide Area Network. Vähävirtaisten laitteiden alueverkko.
LoRa	Esineiden internetin laitteille kehitetty LPWAN-verkko.
NB-IOT	NarrowBand IoT. Tuleva esineiden internetin 4G-verkkostandardi, jonka on tarkoitus toimia käytöstä poistetuilla GSM-taajuuksilla (Collin & Saarelainen 2016, 176).
NB-LTE	NarrowBand LTE. Tuleva esineiden internetin 4G-verkkostandardi, jota kehittävät Nokia, Ericsson ja Intel.
SFR	Special Function Register. Mikroprosessorin toiminnollisuuksien hallintarekisteri.
Sigfox	Esineiden internetin laitteille kehitetty LPWAN-verkko.
SRAM	Static Random Access Memory. Staattinen RAM-muisti.

TDD	Test Driven Development. Testivetoinen kehitys.
TLD	Test Last Development. Perinteinen ohjelmistonkehitysmuoto, jossa testit luodaan ohjelman luonnin jälkeen.
UART	Universal Asynchronous Receiver Transmitter. Yleiskäyttöinen asynkroninen lähetin/vastaanotin.
UNB	Ultra Narrow Band. Teknologia, jolla radiotaajuusalue saadaan jaettua erittäin pieniin osiin (SIGFOX 2016d).
VDDIO	AX8052F143:n käyttöjännite.
VPN	Virtual Private Network. Julkisen verkon yli kulkeva yksityinen verkko.
Weightless	Esineiden internetin laitteille kehitetty LPWAN-verkko, jota hallinnoi etujärjestö Weightless SIG (Collin & Saarelainen 2016, 179–180).
XREG	AX8052:n rekisteri toiminnollisuuksille, jotka eivät mahdu SFR-rekisteriin (ON Semiconductor 2016d, 14).

1 JOHDANTO

Opinnäytetyön aiheena on rajapintojen kehittäminen Sigfox-järjestelmäpiirille. Projektin toimeksiantajayritys on turkulainen sulautettujen järjestelmien ohjelmisto- ja laitteistokehitykseen erikoistunut yritys RCP Software Oy.

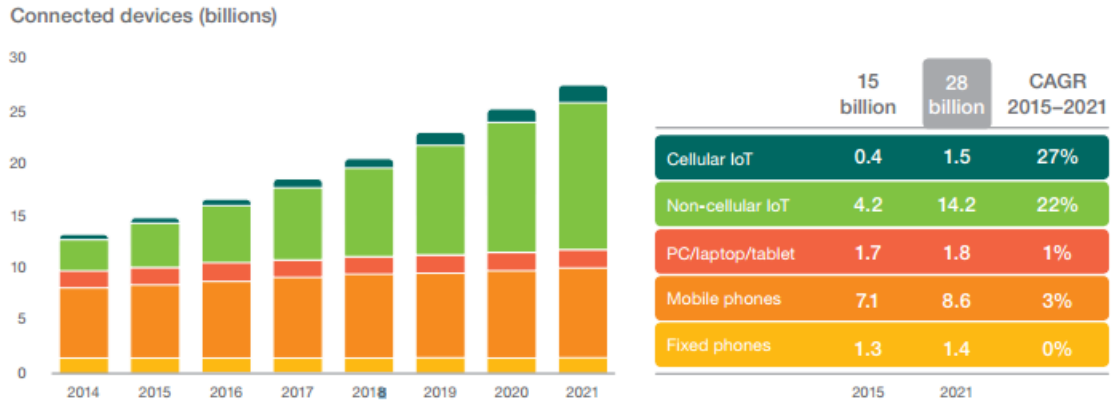
Esineiden internetin laitteiden yhdistämiseen on kehitetty monenlaisia teknologioita. Esineiden internetin yleistyessä myös teknologioiden välinen kilpailu kiristyy. Yksi matalan virrankulutuksen omaavien laitteiden yhdistämiseen kehitetty verkko on ranskalaisen SIGFOXin kehittämä, yrityksen nimeä kantava Sigfox-verkko (SIGFOX 2017b). Suomessa ensimmäiset Sigfox-verkot avattiin syyskuussa 2016 (Connected Finland 2016). Sigfox-verkkoon yhdistettäviä päätelaitteita on saatavilla monilta valmistajilta, ja sellaisen voi myös kehittää itse. RCP Software Oy on kehittänyt oman Sigfox-järjestelmäpiirin, jonka jatkokehittämistä varten tarvitaan toimivat rajapinnat UARTin ja GPIO:n hallinnalle sekä toiminnollisuus piirin hallitsemiseen UARTin välityksellä.

Ohjelmistojen kehitykseen käytetään yhä enemmän ketteriä menetelmiä. Ketterien menetelmien periaatteisiin kuuluu mm. toimivan ohjelmiston jatkuva toimittaminen sekä toimivan ohjelmiston toiminta pääasiallisena edistyksen mittarina (Agile Manifesto 2017). Testivetoinen kehitys on ohjelmistonkehitysmalli, jonka hyödyt tukevat näitä periaatteita. Tässä opinnäytetyössä selvitetään testivetoisen kehityksen toimintamalli ja tavat, joilla testivetoista kehitystä voidaan hyödyntää sulautetun ohjelmiston kehityksessä.

Työn tavoitteena on luoda toimivat rajapinnat Sigfox-järjestelmäpiirille, jotta piiriä pystyttäisiin käyttämään asiakkaille räätälöidyissä ratkaisuissa. Tavoitteena on myös käyttää testivetoista kehitystä rajapintojen kehityksessä ja saada ymmärrys sen tuomista eduista ja haitoista. Alustavien tietojen perusteella oletetaan, että testivetoisen kehitysmallin mukainen kehitystyö tuottaa alusta pitäen toimivampaa ohjelmakoodia, jolloin jälkeenpäin tapahtuvaan virheiden jäljitykseen ei tarvitse kuluttaa yhtä paljon aikaa verrattuna perinteisiin menetelmiin.

2 SIGFOX

Esineiden internet on jo arkipäivää etenkin suurien teollisuuden yritysten osalta (Collin & Saarelainen 2016, 17). Lähivuodet tulevat olemaan merkittäviä esineiden internetin kasvulle. Ericsson arvioi vuonna 2016 esineiden internetin yhdistettyjen laitteiden määrän nousevan vuoden 2015 15 miljardista 28 miljardiin vuoteen 2021 mennessä (Kuvio 1). Esineiden internetin laitteiden ennustetaan ohittavan matkapuhelimet yhdistettyjen laitteiden määrässä vuonna 2018. (Ericsson 2016, 10.) Vaikka esineiden internetin yhdistettyjen laitteiden määrän ennustukset on todettu olevan yliampuvia, voidaan silti olettaa, että kasvu tulee olemaan suurta lähivuosina (IEEE Spectrum 2016). Monia tiedonsiirtoteknologioita ja protokollia on luotu esineiden internetin laitteiden yhdistämiseksi. Sigfox-verkko on yksi useasta esineiden internetin laitteille kehitetyistä verkoista.

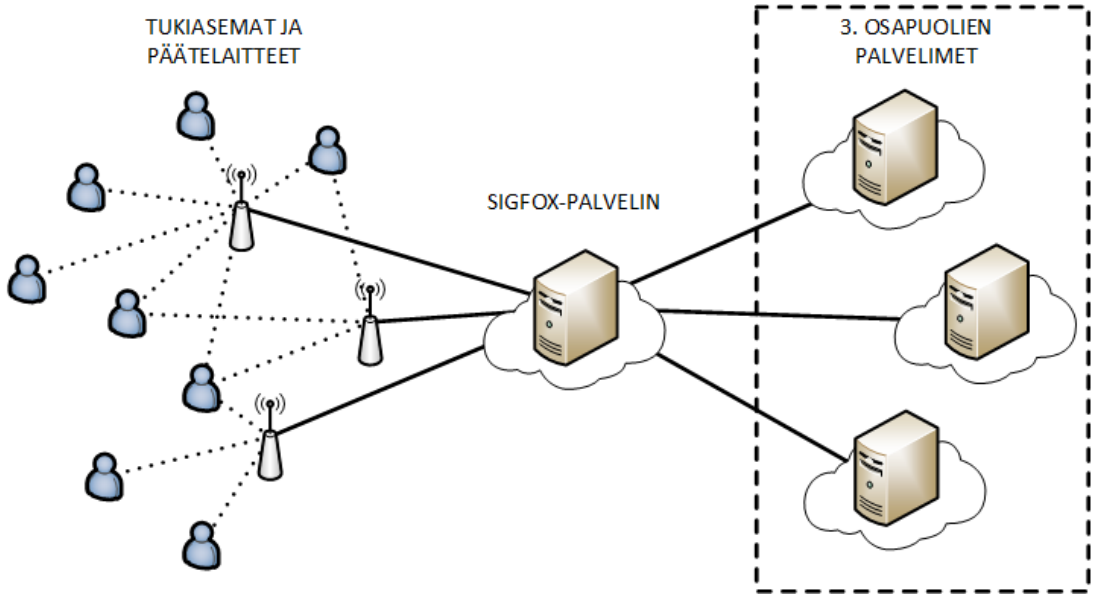


Kuvio 1. Ericssonin arvio esineiden internetin yhdistettyjen laitteiden määrän kehityksestä (Ericsson 2016, 10).

SIGFOX on ranskalaisten Ludovic de Moanin ja Christophe Fourtetin perustama yritys, joka kehittää ja hallinnoi esineiden internetille tarkoitettua Sigfox-verkkoa. SIGFOX perustettiin vuonna 2009, ja ensimmäiset Sigfox-tukiasemat otettiin käyttöön vuonna 2012. Sigfox-verkko on laajentunut vuoteen 2017 mennessä 32 valtioon tavoittaen 512 miljoonaa ihmistä. (SIGFOX 2017b.) Sigfox-verkon avainominaisuuksia ovat matala energiankulutus, kustannustehokkuus ja yhteensopivuus muiden langattomien teknologioiden kanssa (SIGFOX 2016c).

2.1 Sigfox-verkon toiminta

Sigfox verkko on alle 1 GHz:n radiotaajuuksilla toimiva matalan energiakulutuksen alueverkko (LPWAN). Euroopassa Sigfox-verkko toimii 868 MHz:n taajuusalueella suurimman sallitun lähetystehon ollessa 14 dBm. Verkon topologia on tähtimallinen. Verkko koostuu päätelaitteista, tukiasemista, Sigfox-palvelimesta sekä sovelluskohtaisista pilvipalvelimista (Kuva 1). (SIGFOX 2016a.)



Kuva 1. Sigfox-verkon topologia (SIGFOX 2016a).

Kommunikaatio verkossa on kaksisuuntainen. Viestintää rajoittaa päätelaitteilta lähetettävien viestien koko, joka saa olla enintään 12 t. Päätelaitteelle vastaanotettavien viestien koko saa olla enintään 8 t. Yksi päätelaite saa lähettää vuorokauden aikana enintään 140 viestiä. (SIGFOX 2016a.) Sigfox-protokollan viestintä on kehitetty mahdollisimman kevyeksi. Yksi viestipaketti on kooltaan enintään 26 t, kun viestin otsikkotiedot lasketaan mukaan. Lyhyet viestit mahdollistavat paristokäyttöisten päätelaitteiden pitkän toiminta-ajan sekä sallii verkolle enemmän yhdistettyjä laitteita kuin perinteiset viestintäprotokollat. (SIGFOX 2016b.) Sigfox-verkko hyödyntää UNB (Ultra Narrow Band) -radioteknologiaa, jossa radiotaajuudet on jaettu pienempiin kaistoihin mahdollistaen useamman laitteen kommunikoinnin samaan aikaan (SIGFOX 2016d).

Sigfox-päätelaitteen ja tukiaseman välillä ei ole viestien lähetteleyn lisäksi mitään muuta keskustelua tai merkinantoja. Päätelaite päättää viestin lähettämisen ajankohdan sekä

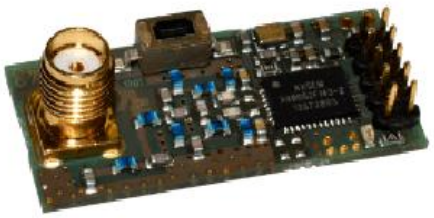
käytettävän taajuuden, joka valitaan pseudo-satunnaisesti. Tukiaseman tehtävä on vastaanottaa viesti ja välittää se Sigfox-palvelimelle, josta se voidaan edelleen välittää kolmansien osapuolien palvelimille. Kaikki Sigfox-verkon viestit kulkevat Sigfox-palvelimen kautta. Viestien lähettäminen päätelaitteelle tapahtuu aina päätelaitteen aloitteesta. Sigfox-protokollalle on määritelty aikaikkunat viestin vastaanottamiselle, jonka aikana päätelaite kuuntelee verkkoa. Päätelaitteen tulee ensin lähettää Sigfox-verkkoon viesti, joka sisältää tiedon halukkuudesta vastaanottaa viesti. Päätelaitteelle lähetettävä viesti voi olla staattinen, jolloin viestiä on mahdollista päivittää API-kutsujen avulla, tai dynaaminen, jolloin viesti lähetetään suoraan sovelluskohtaiselta palvelimelta. (SIGFOX 2017.)

2.2 Tietoturva

Päätelaitteilla on oma salainen avain sekä käyttäjälle näkyvä tunniste viestien yksilöimiseen. Avainta käytetään jokaisessa viestissä digitaalisen allekirjoituksen luomiseksi. Jokaisella lähetetyllä viestillä on yksilöllinen allekirjoitus, jolla lähettäjä tunnistetaan. Allekirjoitukseen on sulautettu järjestysnumero, jolla estetään saman viestin toistaminen. Jokainen viesti voidaan salata sovelluskohtaisesti erillisellä salausteknologialla (SIGFOX 2016c). Viestien vastaanotto on pyritty suunnittelemaan tietoturvallisuuden kannalta lujaksi. Koska päätelaite kuuntelee verkkoa vain silloin ja sillä taajuudella kuin itse päättää, pysyen muuten verkon ulkopuolella, on ulkopuolisten tahojen erittäin hankala lähettää sille viestejä. Sigfox-tukiasemien ja Sigfox-palvelimien välinen yhteys on turvattu salatulla VPN-yhteydellä. Sigfox-palvelimet ovat virtualisoituja ja sijaitsevat yksityisesti ylläpidetyillä palvelimilla. Palvelimien sijainnit on jaoteltu useisiin tietokeskuksiin. Kolmansien osapuolien palvelimet ovat yhteydessä Sigfox-palvelimiin HTTPS-yhteydellä. (SIGFOX 2017c.)

2.3 Päätelaitteet

Sigfox-päätelaitteiden valmistus on täysin avointa sekä rojaltivapaata. Monet puolijohdevalmistajat tarjoavat erilaisia päätelaiteratkaisuja kokonaisista modeemeista yksittäisiin järjestelmäsiruihin (Kuva 2). (Briodagh 2016.)



Kuva 2. ON Semiconductorin tarjoama kaupallinen Sigfox-modeemi AX-SFEU (ON Semiconductor 2017).

Suosituille kehitysalustoille kuten Arduinolle ja Raspberry Pille on saatavilla Sigfox-moduuleja, joilla Sigfox-verkon käyttäminen on helppo aloittaa. Arduino on julkaissut Arduino Zero -kehityspiiriin pohjautuvan Arduino MKRFOX 1200 -kehityspiirin, jossa on valmius Sigfox-verkolle (Kuva 3). Piirin hinta on 35 €. (Arduino 2017.)

Kuva 3. Arduino MKRFOX 1200 -kehityspiiri (Arduino 2017).

Sigfox-verkon vahvuuksiin kuuluu kustannustehokkuus, mikä näkyy päätelaitteiden hinnoissa. SIGFOX auttaa puolijohdevalmistajia piisirujen optimoinnissa, jotta kustannukset saataisiin mahdollisimman alas. Valmiita Sigfox-verkkoon liitettäviä piirejä on saatavilla alle 3 €:n hintaan. Avoimuus piirilevyjen suunnittelussa mahdollistaa myös muiden teknologioiden sisällyttämisen Sigfoxin rinnalle. (Briodagh 2016.)

Päätelaitteen sertifiointi

Päätelaitteen voi valmistaa myös itse, jolloin se tarvitsee sertifioida. Jokaisen Sigfox-verkkoon kytkettävän laitteen tulee olla Sigfox Ready -sertifioitu (Kuva 4). Sertifiointiprosessi varmistaa laitteen turvallisuuden sekä oikeanlaisen toiminnan Sigfox-verkossa. (SIGFOX 2017e.)

Kuva 4. Sigfox Ready -sertifioinnin tunnus (SIGFOX 2016e).

Sigfox Ready -sertifikaatti ei kata paikallisia säädöksiä. Prototyyppejä ei voi sertifioida, vaan laitteen tulee olla valmis kaupallistettava tuote. Sertifikaatin myöntää SIGFOX. Testit suoritetaan SIGFOXin valitsemissa laboratorioissa. Maakohtaiset operaattorit ovat vastuussa maan Sigfox-ekosysteemin hallinnasta, sisältäen myös sertifiointiprosessin. (SIGFOX 2017e.)

2.4 Markkina-asema

Sigfoxin merkittävimpiä kilpailijoita ovat LPWAN-verkot LoRaWAN ja Weightless. Molemmat kilpailijat on suunniteltu Sigfox-verkon tapaan vastaamaan tarpeeseen laajan peittoalueen ja matalan energiankulutuksen omaavista langattoman tiedonsiirron laitteista. Kaikki mainitut verkot käyttävät samoja lisensoimattomia taajuuksia, jotka voivat tulevaisuudessa mahdollisesti ruuhkautua, kun kyseisiä verkkoja hyödyntävien laitteiden määrä kasvaa. Tulevia kilpailijoita ovat NB-IOT (NarrowBand IoT) sekä NB-LTE (NarrowBand LTE), jotka toimivat GSM-taajuuksilla. (Collin & Saarelainen 2016, 176–178.)

Sigfoxin erottaa kilpailijoista käyttöönoton helppous ja yksinkertainen ekosysteemi. Sigfox toimii maailmanlaajuisesti samalla tavalla kilpailijoiden tukeutuessa paikallisiin ratkaisuihin. Kilpailijoilla on etu Sigfoxiin nähden avoimuudessa. Molemmat LoRa ja Weightless ovat avoimia protokollia. Kuka vaan voi pystyttää kyseisen verkon, kunhan käyttää tarvittavia komponentteja. SIGFOX puolestaan hallinnoi koko maailmanlaajuista Sigfox-verkkoaan maakohtaisten operaattoreiden avustuksella.

Kuva 5. Sigfox-verkon peittävyys Suomessa 21.4.2017 (SIGFOX 2017d).

Suomessa ensimmäiset Sigfox-verkot avattiin syyskuussa 2016. Verkon operaattorina toimii Connected Finland. Sigfox-verkko kattaa tällä hetkellä suurimmat kaupungit (Kuva 5). Verkon on suunniteltu peittävän vuoden 2017 kesään mennessä 85 % Suomen asukkaista. (Connected Finland 2016.)

3 TESTIVETOINEN KEHITYS

TDD (Test Driven Development), eli testivetoinen kehitys, on ohjelmistonkehitystekniikka, jossa ohjelmistoa kehitetään luomalla testitapaukset ennen ohjelmakoodia. Automatisoidut testit ajavat ohjelman toteutusta ja edistävät ohjelmamoduulien keskinäistä riippumattomuutta (Palermo 2006). TDD:ssä testitapaukset ovat yhtä arvokkaita ohjelmakoodin kanssa. Ohjelmakoodin kasvaessa vanhat testitapaukset pitävät huolen siitä, etteivät uudet ominaisuudet riko ohjelmakokonaisuuden toimintaa. TDD on vaihtoehtoinen kehitysmuoto perinteiselle TLD:lle (Test Last Development), jossa ohjelmakoodi luodaan ensin ja testataan jälkeenpäin.

3.1 Hyödyt ja heikkoudet

TDD:n hyötyjä kuulutetaan monessa paikassa, mutta kehitysmallista löytyy myös heikkouksia. Grenningin (2011, 8–9) mukaan TDD:n tuomia hyötyjä ovat

- virheiden määrän pienentyminen
- virheiden jäljitykseen ja korjaamiseen kuluneen ajan pienentyminen
- sivuvaikutuksina ilmentyvien vikojen vähentyminen
- yksikkötestien toimiminen dokumentaationa
- varmuus ohjelman toiminnasta
- ohjelman parempi rakenne
- testien toiminta etenemisen mittarina.

TDD:ssä on myös heikkouksia verrattuna perinteiseen TLD:hen (Hill 2015):

- Testien luominen vie aikaa kehityksen alussa, jolloin kehittäminen voi tuntua hitaalta.
- Keskittymällä pieniin yksityiskohtiin miettimättä tulevia tarpeita voi johtaa suuriin refaktorointeihin tulevaisuudessa.
- Hyvien testien kirjoittaminen on vaikeaa.
- Testiympäristön ylläpitäminen vie aikaa ja sitä pitää konfiguroida jatkuvasti suurimman mahdollisimman hyödyn takaamiseksi.
- Jos ohjelman suunnitelma muuttuu nopein aikavälein, pitää testitkin muuttaa suunnitelmaa vastaaviksi.

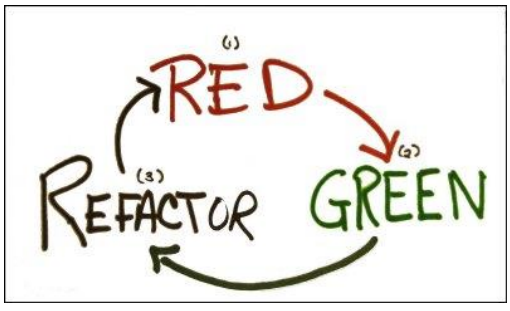
TDD ei sovi jokaiselle projektille. TDD:n käytön tarvetta tulisi pohtia jokaiselle projektille erikseen. Etenkin projekteihin, joissa työskennellään vanhojen ohjelmien kanssa, voi olla vaikea sisällyttää TDD:tä. (Hill 2015.)

TDD vie enemmän aikaa

Yleinen oletus on, että TDD vie enemmän aikaa, sillä ohjelmakoodin lisäksi kehittäjän pitää luoda jatkuvasti testikoodia. Väite on totta, jos oletetaan, että vaihtoehtona on kehitysmuoto, jossa kehittäjä luo jatkuvasti virheetöntä ohjelmakoodia. Todellisuutta kuitenkin on, että virheet ohjelmakoodissa ovat arkipäivää ja ne tulee korjata huolimatta siitä, käytetäänkö kehityksessä TDD:tä vai ei. TDD:n tuoma etu on se, että virheet huomataan nopeasti. TLD:ssä virheitä voidaan joutua jäljittämään ajallisesti pitkään. (Grenning 2011, 93.)

3.2 Kehityksen vaiheet

TDD:ssä kehitys etenee pienin askelin. Jokaisesta halutusta ominaisuudesta luodaan ensin yksikkötesti, jonka jälkeen luodaan testiä vastaava ohjelmakoodi. Ohjelmakoodin läpäistessä testin tuotettu koodi refaktoroidaan, jonka jälkeen siirrytään seuraavaan ominaisuuden testin luomiseen. Kuvattu järjestys on TDD:n mikrosykli. Sykliä toistetaan, kunnes ohjelman vaadittu kokonaisuus on valmis.



Kuva 6. TDD:n mikrosykli punainen-vihreä-suunnittele (Martin 2014).

TDD:n mikrosykliä kuvataan termillä punainen-vihreä-suunnittele (red-green-refactor) (Kuva 6). Termi syntyi Java-kehittäjien keskuudessa. Java-ohjelmien testaamiseen käytetty JUnit-testikehys tarjoaa graafisen esityksen testien tuloksista. Punainen kuvaa epäonnistunutta testiä ja vihreä kaikkien testien läpäisyä. TDD:n malliin kuuluu, että testit

tulee aina ajaa ennen ohjelmakoodin toteuttamista. Näin jokainen kehityssykli alkaa aina punaisesta. Testien onnistuessa tuotettu koodi ”suunnitellaan” eli refaktoroidaan. (Grenning 2011, 9.) Kun TDD:tä käytetään tehokkaasti, yksi mikrosykli kestää normaalisti muutamia minuutteja (Martin 2014).

3.3 TDD:n kolme lakia

Martin (2005) tiivistää TDD:n ytimekkäästi kolmeen lakiin:

- Älä kirjoita ohjelmakoodia, ellei sen tarkoitus ole saada epäonnistuvaa testiä onnistumaan.
- Älä kirjoita enempää yksikkötestejä kuin on tarpeellista epäonnistumisen saavuttamiseksi.
- Älä kirjoita enempää ohjelmakoodia kuin on tarpeellista saadaksesi yhden yksikkötestin onnistumaan.

Ohjelmaa luotaessa kokonaiskuvan ajattelu on väistämätöntä ja tarpeellistakin, jolloin saattaa syntyä halu oikaista lakien noudattamisessa. TDD:n kolmen lain seuraaminen kurinalaisesti johtaa kuitenkin kattaviin testeihin ja läpikotaisin testattuun ohjelmakoodiin. (Grenning 2011, 36.)

3.4 Sulautetun ohjelmiston kehitys TDD:llä

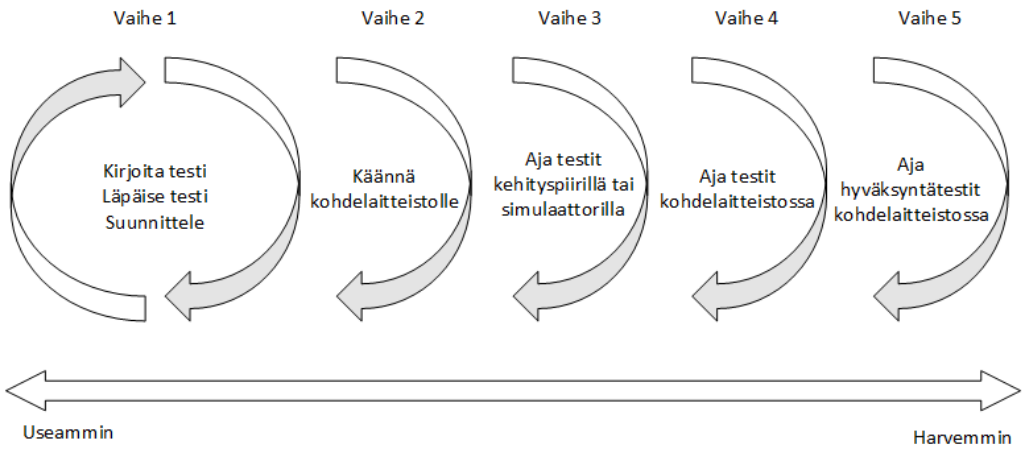
Sulautettujen ohjelmistojen kehitys eroaa muista ohjelmistoalustoista merkittävästi. Sulautettu ohjelmisto on riippuvainen käytetystä laitteistosta. Siitä huolimatta TDD tarjoaa samat edut myös sulautettujen ohjelmistojen kehitykseen. Riippuvuudet ovat haaste myös muiden alustojen kehityksessä. TDD:n näkökulmasta sulautetun ohjelmiston riippuvuus laitteistosta ei juurikaan eroa riippuvuudesta tietokannan kanssa. Ideana on ohjelmiston kehittäminen irrallaan laitteistosta erillisessä testiympäristössä, jossa tarvittavat riippuvuudet voidaan virtualisoida. (Grenning 2011, 10.) TDD:n tuomia hyötyjä sulautetun ohjelmiston kehityksessä ovat (Grenning 2011, 10):

- riskin vähentäminen vahvistamalla ohjelmakoodi irrallaan laitteistosta, ennen kuin laitteisto on valmis tai kun laitteisto on liian kallis
- kääntämiseen, linkitykseen ja ohjelman lataamiseen käytettyjen kertojen määrän vähentyminen virheidenkorjauksessa

- vähentynyt virheenjäljitysaika kohdelaitteistossa
- laitteiston ja ohjelmiston vuorovaikutusongelmien eristäminen etukäteen, kun laitteiston käyttäytyminen on mallinnettu testeissä
- kehittyneempi ohjelman suunnittelu, kun moduulit pysyvät riippumattomina toisistaan ja laitteistosta.

Sulautettujen ohjelmistojen kehityksessä kehityssykliin lisätään vaiheet, joilla varmistetaan tuotetun ohjelmakoodin toiminta kohdelaitteistossa (Kuva 7). Sulautetun TDD:n vaiheita on 5 (Grenning 2011, 80–83):

1. TDD:n mikrosykli. Ohjelmakoodia tuotetaan erillisessä testiympäristössä. Tuotettu koodi on mahdollisimman laitteistoriippumatonta.
2. Kääntäjän yhteensopivuuden tarkistus. Ajoittain ohjelmakoodi käännetään kohdelaitteiston kääntäjällä, jolloin voidaan huomata mahdolliset yhteensopivuusongelmat.
3. Yksikkötestit kehityspiirillä. On olemassa riski, että ohjelmakoodi käyttäytyy eri tavalla kohdelaitteen prosessorilla kuin testiympäristössä. Yksikkötestaaminen kehityspiirillä tai paremmassa tapauksessa itse kohdelaitteistolla auttaa löytämään nämä erot.
4. Yksikkötestit kohdelaitteistolla. Vaiheen 3 tavoin sillä lisäyksellä, että tässä vaiheessa ajetaan myös laitteistokohtaisia testejä.
5. Hyväksyntätestit kohdelaitteistolla. Varmistetaan, että tuotetun ohjelman toiminnollisuudet toimivat ajamalla automatisoituja ja manuaalisia hyväksyntätestejä. Kaikki laitteistokohtaiset ohjelmakoodin osat, joita ei voida testata automatisoidusti, testataan manuaalisesti.



Kuva 7. Sulautetun ohjelmiston TDD-sykli (Grenning 2011, 82).

Projektin laitteisto ei ole aina valmiina tai muuten käytettävissä, jolloin vaiheita 4 ja 5 ei voida suorittaa. Kun laitteisto on saatavilla, vaiheen 3 voi jättää pois syklistä. (Grenning 2011, 83.)

Kohdelaitteiston rajallinen muisti

Sulautettujen ohjelmistojen kohdelaitteistoilla on yleensä rajallinen määrä muistia käytettävänään. TDD:n käyttämiseen tarvittavat testit sen sijaat vievät tilaa muistissa ohjelmakoodin lisäksi. Ongelmaa voidaan kiertää mm. tekemällä kehitysversio kohdelaitteistosta, jossa on riittävä määrä muistia ohjelmakoodille sekä testitapauksille. Testit voidaan myös pilkkoa sopiviin osiin, jotka mahtuvat kohdelaitteelle, ja ajaa sitten erikseen. (Grenning 2011, 99–100.)

3.5 TDD:n soveltaminen kehitysprojektissa

Tämän opinnäytetyön projektityössä sovelletaan testivetoista kehitystä Sigfox-järjestelmäpiirin kehityksessä. Koska TDD:n oikeaoppinen käyttö ja sen tuomien hyötyjen saavuttaminen vaativat pitkäaikaista ja kurinalaista opettelua, ei tämän projektin osalta TDD:n oppeja pyritä noudattamaan kirjaimellisesti. Projektityössä käytetään testiympäristönä tavallista työasemaympäristöä, ja testit luodaan C-komentoriviohjelmina. Opinnäytetyöprojektin alussa arvioitiin, että testikehyksen käytön opettelu ei ole tarpeen tämän projektin osalta.

4 KEHITYSPROJEKTIN ALOITUS

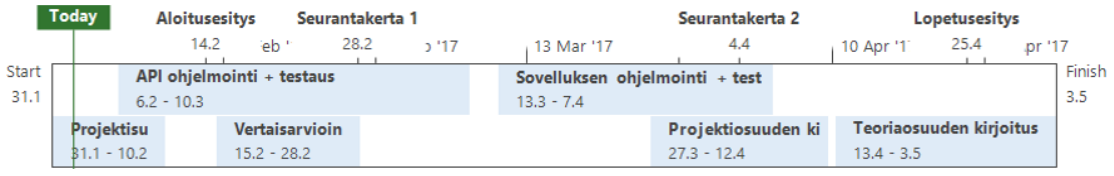
Projekti aloitettiin koostamalla vaatimusmäärittely yhdessä toimeksiantajan kanssa, luomalla alustava aikataulu, kehitysympäristöön ja komponentteihin tutustumalla sekä kartoittamalla järjestelmäpiirille saatavilla olevia ohjelmakirjastoja.

4.1 Vaatimusmäärittely

Kehitettävän järjestelmäpiirin GPIO-porttien tulisi olla ohjelmoitavissa eri asiakkaiden tarpeiden mukaisesti. UARTin välityksellä lähetettävien komentojen avulla järjestelmäpiiriä tulisi pystyä hallinnoimaan mahdollisimman laajasti. UART-komentojen tulee päättyä rivinvaihtomerkkiin, ja komentojen tyyli voidaan määritellä itse. Vaadittuja komentoja ovat järjestelmäpiirin asetuksien lukeminen ja muokkaaminen, Sigfox-tunnisteiden, lämpötilan ja käyttöjännitteen lukeminen, Sigfox-sertifiointiin tarvittavien testien ajaminen sekä Sigfox-viestien lähettäminen ja vastaanottaminen. Piirin tulisi olla ohjelmoitavissa niin, että sitä voidaan käyttää itsenäisenä esineiden internetin laitteena, tai vaihtoehtoisesti pelkkänä Sigfox-verkon modeemina, jolloin se olisi kytketty UARTilla erilliseen mikro-ohjaimeen.

4.2 Aikataulu

Opinnäytetyön tekemiseen kokonaisuudessaan varattiin aikaa noin 3 kk. Suurin osa ajasta varattiin projektin työstämiselle. Aikataulu suunniteltiin siten, että projektin työstämisen aikana kirjoitettu dokumentaatio toimii pohjana opinnäytetyön projektiosuuden kirjoittamisessa. Teoriaosuuden lähdeaineistoon tutustutaan projektin työstämisen ohessa, ja itse teoriaosuuden kirjoitus tapahtuisi sitten, kun projektiosuuden kirjoitus on valmis. Alustavaan aikatauluun on sisällytetty myös muut opinnäytetyön työstämisen aikana suoritettavat tehtävät (Kuva 8).



Kuva 8. Opinnäytetyöprojektin alustava aikataulu.

4.3 Projektinhallinta ja ohjelmistokehitysmenetelmät

Projekti toteutettiin itsenäisesti opinnäytetyöprojektina. Projektinhallinnan komponentteina koko projektin ajan ylläpidettiin projektipäiväkirjaa sekä tehtävälistaa, jota päivitettiin projektin edetessä. Tehtävälistatyökaluna käytettiin Trelloa, joka on www-pohjainen projektinhallintatyökalu.

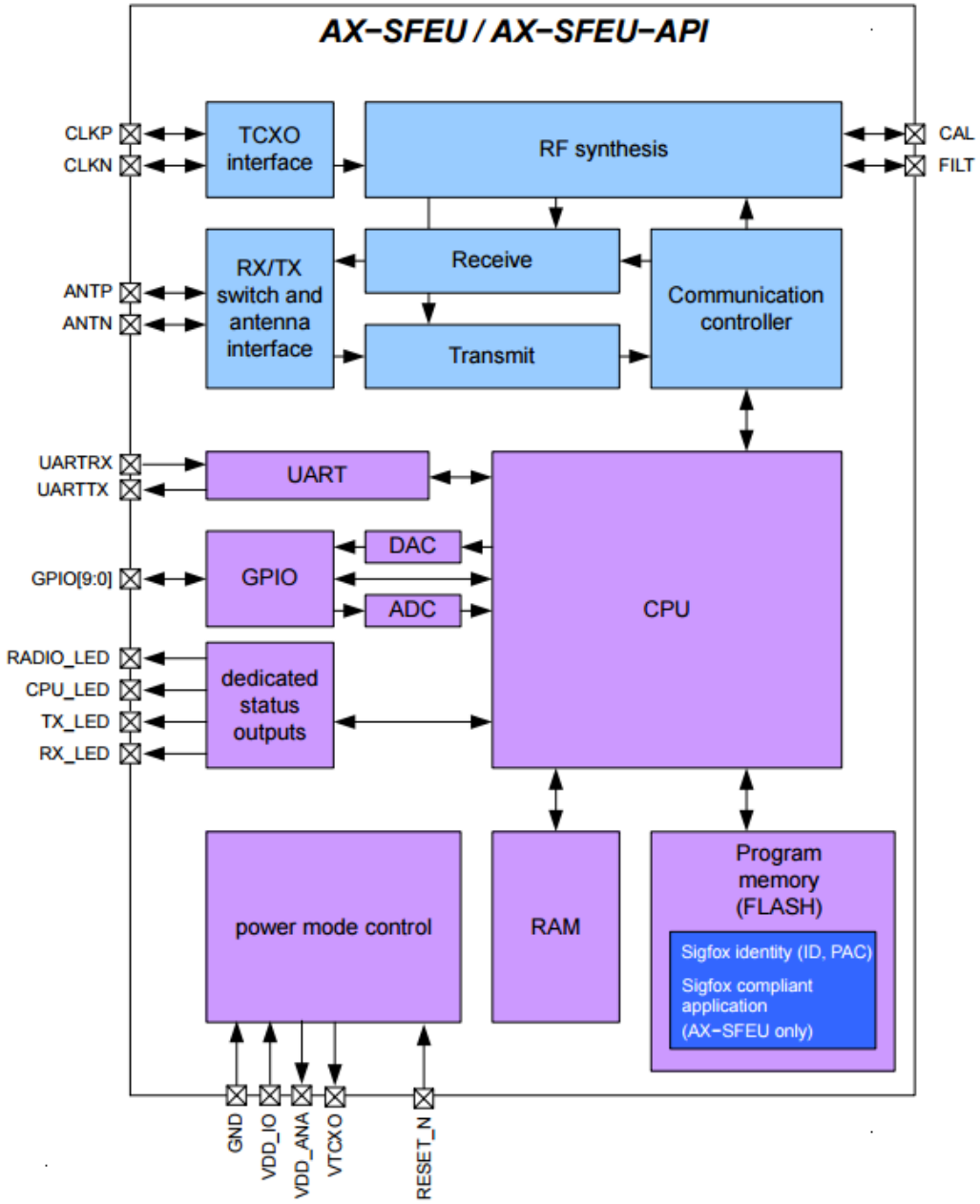
Ohjelmistokehitysmallina sovellettiin ketteriä menetelmiä yhdessä testivetoisen kehityksen kanssa. Ketterien menetelmien tapaan alkuvaiheen suunnittelu jätettiin vähemmälle, ja pääpaino kehityksessä oli toimivien kirjastojen kehityksessä. Kirjastojen arkkitehtuuria pystyttiin muuttamaan kehityksen aikana helposti ketterien menetelmien tuoman joustavuuden ansiosta. Testivetoinen kehitys otettiin käyttöön komentorajapinnan kehityksessä.

4.4 Kehitysympäristö

Projektin työkaluketjuun kuuluu järjestelmäpiirin mikroprosessori, mikroprosessorin virheenjäljittäjäpiiri, USB-UART-muuntaja, Sigfox-järjestelmäpiirin ohjelmointiympäristön komponentit virheenjäljittäjäsovellus, kääntäjä ja tekstinkäsittelyohjelma sekä testien ohjelmointiympäristön komponentit.

Mikro-ohjain

Kehitettävän Sigfox-järjestelmäpiirin mikro-ohjaimena toimii ON Semiconductorin AX-SFEU-API (Kuvio 3). Käytännössä AX-SFEU-API on ON Semiconductorin matalavirtainen radiotaajuusmikro-ohjain AX8052F143. AX-SFEU-API:hin on tehtaalla kalibroitu Sigfox-verkolle tarvittavat asetukset ja tunnisteet, ja sitä on tarkoitus käyttää yhdessä ON Semiconductorin tarjoaman ohjelmointirajapinnan kanssa (ON Semiconductor 2016a, 1).



Kuvio 2. AX-SFEU-API:n lohkokaavio (ON Semiconductor 2016e, 2).

AX8052F143-mikro-ohjain on erittäin matalavirtainen radiotaajuusmikro-ohjain alle 1 GHz:n taajuuksille. Ohjain toimii 1,8 V – 3,6 V käyttöjännitteellä, sisältää 64 kt ohjelmoitavaa flash-muistia, joista AX-SFEU-API:ssä 1 kt on varattu Sigfox-piirikohtaiselle tehdaskalibraatiotiedolle, 8 kt ROM-muistia sekä 8,26 kt RAM-muistia. RAM rakentuu kahdesta 4 kt:n SRAM-muistilohkosta sekä yhdestä 256 t:n IRAM-muistilohkosta. Mo-

lemmat SRAM-muistilohkot voidaan sammuttaa tai säilyttää yksilöllisesti, kun mikro-ohjain asetetaan virransäästötilaan. IRAM-muisti säilytetään aina virransäästötilassa. (ON Semiconductor 2015, 18–19.)

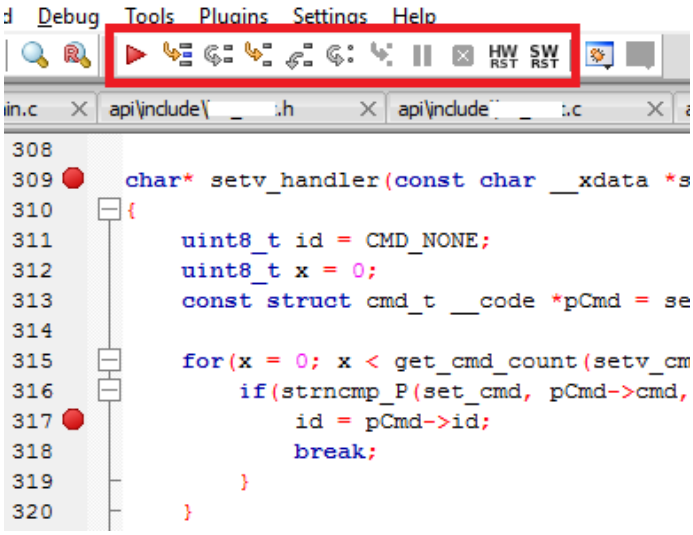
Projektin kannalta tärkeitä ominaisuuksia ovat myös 2 herätysajastinpiiriä, lämpötilasensori, 2 UART-piiriä, 3 yleiseen käyttöön tarkoitettua ajastinpiiriä sekä 24 GPIO-porttia. (ON Semiconductor 2016d, 1.)

Virheenjäljittäjä

Projektissa käytettiin AX8052-mikro-ohjaimelle yhteensopivaa AXSEM AXDBG -virheenjäljittäjäpiiriä. Virheenjäljittäjäpiirillä ja Sigfox-järjestelmäpiirillä on yhtenevät virheenjäljitysliittimet, jotka yhdistetään toisiinsa nauhakaapelilla. Sigfox-järjestelmäpiiri saa virheenjäljityspiiriltä käyttöjännitteen ja mahdollisuuden lähettää viestejä työasemalle DebugLink-protokollalla. Virheenjäljittäjäpiiri yhdistetään työasemaan USB-kaapelilla. Virheenjäljittäjäpiirillä voidaan mm. ladata ohjelmia mikro-ohjaimelle, ajaa Sigfox-piirin mikro-ohjainta virheenjäljitystilassa ja lukea mikro-ohjaimen muistilohkoja. Virheenjäljittäjäpiiriä voidaan hallinnoida Windows-työasemalta AXSDB-komentoriviohjelman kanssa.

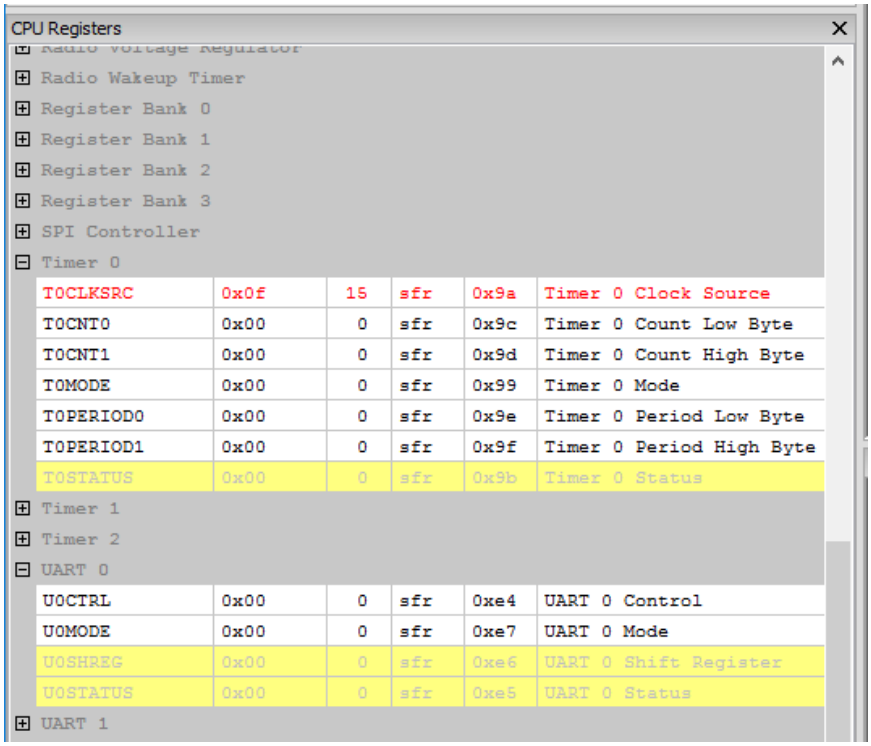
Ohjelmointiympäristöt

Ohjelmointiympäristöksi ON Semiconductor suosittelee integroitua ohjelmointiympäristöä AxCodeBlocksia (ON Semiconductor 2016a, 1). AxCodeBlocks rakentuu nimensä mukaisesti avoimen lähdekoodin CodeBlocks-ohjelmointiympäristön päälle. AxCodeBlocksista on karsittu epäolennaiset ominaisuudet, ja lisätty virheenjäljityspiirin kanssa toimiva virheenjäljitysohjelma AXSDB. AxCodeBlocksin kanssa pystytään asettamaan ohjelmakoodiin pysäytyskäskyjä sekä ajamaan mikro-ohjainta virheenjäljitystilassa (Kuva 9).



Kuva 9. Kuvankaappaus ohjelmointiympäristöstä, jossa ohjelmakoodiin asetettu pysäytyskohtia. Kuvasta korostettu virheenjäljitystilan hallintapainikkeet.

AxCodeBlocksilla pystyy myös tarkastelemaan helposti mikro-ohjaimen rekistereiden arvoja ja muuttamaan niitä, kun mikro-ohjain on pysäytettynä virheenjäljitystilassa (Kuva 10).



Kuva 10. Kuvankaappaus ohjelmointiympäristön rekisterinäkymästä.

Kääntäjäksi ON Semiconductor suosittelee IAR Embedded Workbench for 8051 -ohjelmistopakettia (ON Semiconductor 2016a, 1). IAR:n ohjelmistopaketti sisältää AX8052-perheen mikro-ohjaimille C-kääntäjän, linkitysohjelman, virheenjäljitysohjelman sekä integroidun kehitysympäristön. Tässä projektissa käytetään IAR:n tarjonnasta C-kääntäjää ja linkitysohjelmaa, jotka toimivat sulautettuna AxCodeBlocks-kehitysympäristön kanssa.

Testien luomiseen ei käytetty mitään valmista ohjelmakehystä, vaan testit luotiin tavallisena komentoriviohjelmana. Ohjelmien kääntämiseen käytettiin GCC:n C-kääntäjää yhdessä CodeBlocks-ohjelmointiympäristön kanssa.

Muut työkalut

UART-viestintään käytettiin USB-UART-muuntajapiiriä sekä PuTTY-terminaaliohjelmaa. Työaseman käyttöjärjestelmänä toimi 64-bittinen Windows 10 Pro.

4.5 Kirjastojen kartoitus

Ennen ohjelmointityön aloittamista oli selvitettävä jo olemassa olevat kirjastot. ON Semiconductorin esimerkkisovellusta tutkimalla huomattiin, että sovellus käyttää LibMF-kirjastoja, joihin sisältyi mm. UART-kirjastot. On Semiconductorin web-sivustolta löydettiin tarvittavat dokumentaatiot rajapintojen kehitystä varten.

UART-kirjastoissa on funktiot UARTien alustuksille, erityyppisten muuttujien lähetyksille sekä UARTien FIFO-rekistereiden käsittelyille. Merkkien vastaanottamiseen on vain yksi funktio, joka vastaanottaa yhden merkin kerrallaan. GPIO-kirjastoa ei LibMF-kirjastoista löydetty. ON Semiconductorin www-sivuilta saatiin LibMF-kirjastojen dokumentaation lisäksi AX8052-mikro-ohjainten ohjelmointiopas, AX8052F143-mikro-ohjaimen tietolomake sekä virheenjäljittäjäohjelman käyttöopas.

Kirjastokartoituksen jälkeen voitiin todeta, että erillinen UART-kirjasto on tarpeellinen, jotta viestien vastaanottaminen tapahtuisi halutulla tavalla. Myös GPIO-kirjasto tarvittaisiin, sillä sellaista ei LibMF-kirjastoista löytynyt.

4.6 Kehitysympäristöön tutustuminen

ON Semiconductor suosittelee kehittämään sovellukset muokkaamalla heidän tarjoamaansa esimerkkisovellusta (ON Semiconductor 2016a, 1). Esimerkkisovelluksesta näkee, miten tarvittavat pinot alustetaan, ja millainen pääohjelman rakenteen tulee olla. Esimerkkisovelluksella on mahdollista lähettää ja vastaanottaa yksinkertaisia Sigfox-viestejä. Komennot esimerkkisovelluksessa toimivat DebugLink-protokollalla. AxCodeBlocks sisältää oman DebugLink-terminaalinäkymän, jossa DebugLink-viestejä voidaan tulostaa ja lähettää.

AX-SFEU-API käyttää Sigfox-protokollan hallintaan ON Semiconductorin omaa ohjelmointirajapintaa. ON Semiconductorin rajapinta rakentuu Sigfox-ohjelmointirajapinnan päälle tehden Sigfox-pinon käytöstä helpompaa kehittäjälle. Esimerkkisovellukseen, ON Semiconductorin ohjelmointirajapintaan sekä mikro-ohjaimen arkkitehtuuriin tutustuttiin ennen ohjelmointityön aloittamista. Kokeilemalla eri toiminnollisuuksien liittämistä esimerkkisovellukseen auttoi järjestelmän kokonaiskuvan hahmottamisessa.

5 UART-RAJAPINNAN KEHITYS

5.1 Kehityksen tavoitteet

UART-rajapinnan kehityksen tavoitteina oli selvittää UARTin toiminta AX-SFEU-API-mikro-ohjaimella ja luoda rajapinta, jonka kanssa onnistuu vaatimusmäärittelyn mukainen kommunikaatio UART-protokollalla. UART-rajapinnan tulee tunnistaa viestien päättyminen rivinvaihtomerkistä. Lisäksi UARTin kautta tulisi pystyä herättää mikro-ohjain unitilasta.

5.2 Kehityksen aloitus

UART-kirjaston kehitys aloitettiin selvittämällä UART-porttien sijainnit Sigfox-piirillä. Toimeksiantajayritykseltä saatiin kytkentä- ja ladontakaaviot piiristä, joiden avulla selvitettiin UART-porttien sijainti. Kytkennästä huomattiin myös, että AX8052F143-mikro-ohjaimen sisältämistä UART-piireistä vain toinen (UART0) oli saatavilla UART-protokollan käyttöön. Esimerkkiprojekti kopioitiin uudeksi projektiksi, jonka päälle UART-rajapintaa alettiin kehittää.

Työaseman ja Sigfox-piirin välille pyrittiin luomaan yksinkertainen yhteys UART-protokollaa käyttäen. Esimerkkiprojektin DebugLink-komennot muunnettiin suoraan UART-komennoiksi käyttämällä LibMF:n UART-kirjastoja. Muunnos oli yksinkertainen, sillä kyseisien kirjastojen funktiot ovat tyyliltään identtiset.

UARTin ajastin alustetaan valitsemalla käytettävä kellosignaalin lähde sekä haluttu tiedonsiirtonopeus. UART käyttää ajastinpiirinään yhtä kolmesta mikro-ohjaimen yleiseen käyttöön tarkoitetusta ajastinpiiristä. Kellolähde valitaan saatavilla olevista oskillaattoreista (Tee taulukko). Lisäksi käytettävän UART-piirin asetukset alustetaan kyseisen UARTin omalla alustusfunktiolla, jolla määritellään käytettävä ajastin, sanan pituus, eli bittien määrä aloitus- ja lopetusbittien välissä sekä lopetusbittien määrä. (ON Semiconductor 2016c.)

Yksi ajastinpiiri alustettiin käyttämään FRCOSC-oskillaattoria kellolähteenä, ja tiedonsiirtonopeus asetettiin 9 600 baudiin. UART0:lle asetettiin ajastinpiiriksi kyseinen ajastin,

sananpituudeksi 8 ja lopetusbittien määräksi 1. Tämän jälkeen UART-yhteyttä testattiin terminaaliohjelmalla, mutta yhteys ei toiminut.

Ongelmaa UART-yhteyden muodostamisessa etsittiin tarkistamalla alustusarvot, kokeilemalla alustusta eri arvoilla ja käyttämällä eri ajastinta ja kellolähteitä. Mikään edellä mainituista toimenpiteistä ei auttanut. Mikro-ohjaimen GPIO-porttien ominaisuuksia tutkimalla selvisi, että UARTin käyttämät portit pitää konfiguroida erikseen UART-protokollaa varten. Tästä ei ollut minkäänlaista mainintaa LibMF:n dokumentaatiossa, joten se oli jäänyt huomaamatta.

UART0 toimii mikro-ohjaimen porteilla PB4 (lähetys) ja PB5 (vastaanotto). Tarvittavat rekisterimuutokset on kuvattu alla (Ohjelmakoodi 1). GPIO-rajapinnan kehitys -osiossa kerrotaan tarkemmin GPIO-porttien konfiguroinnista.

Ohjelmakoodi 1. UART0:n GPIO-porttien konfiguraatio.

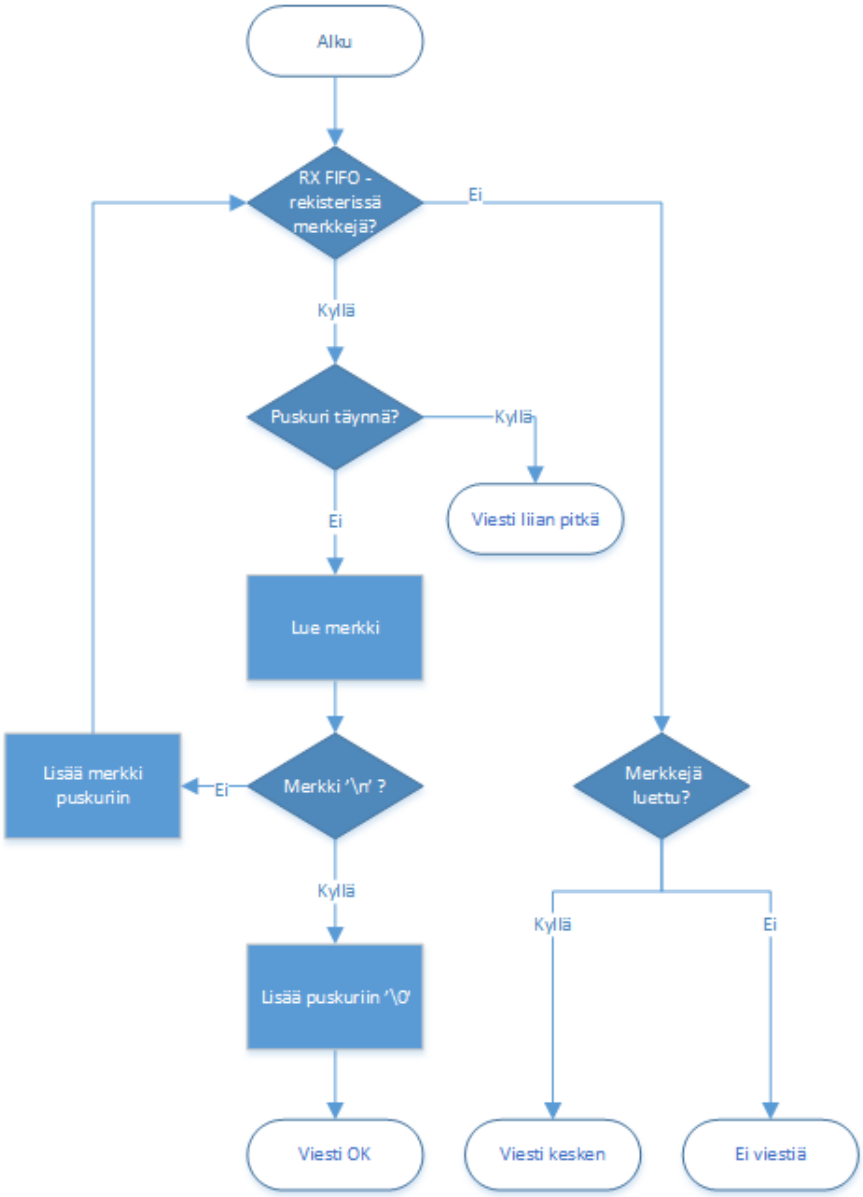
```
/** UART_RX configuration */
// Set PB5 as U0RX
PINSEL &= ~(0x01);

/** UART_TX configuration*/
// Set port direction as output
DIRB |= PORT_PB4;
// Use alternate function (U0TX)
PALTB |= PORT_PB4;
```

Konfiguroinnin jälkeen UART-yhteyden muodostaminen onnistui ja vaatimusmäärittelyn mukaisen toiminnollisuuden kehittäminen aloitettiin.

5.3 Ensimmäinen versio UART-kirjastosta

UARTin tärkein tarvittava ominaisuus oli merkkijonon lukeminen. Merkkijonon lukemiseksi luotiin `UART_readline`-funktio, jossa käytettiin LibMF:n UART-kirjastosta löytyviä apufunktioita. Funktion kutsuminen on sovelluksesta riippuvainen. Rajapinnan kehityksessä luotiin esimerkkisovellus, jossa pääsilmukassa tarkistetaan, onko UARTin FIFO-rekisterissä merkkejä. Kun merkki on saatavilla, siirrytään `UART_readline`-funktioon.

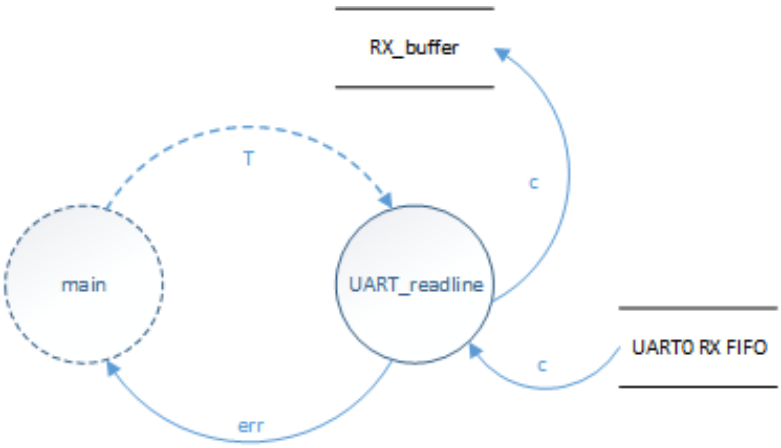
Ensimmäisen version toteutuksessa tavoitteena oli saada merkkijono luettua puuttumatta liikaa toteutustapaan tai ohjelmakoodin selkeyteen. `UART_readline`-funktio toteutettiin niin, että viestien lukeminen ei tuki pääohjelman suoritusta (Kuvio 4).



Kuvio 3. Vuokaavio `UART_readline`-funktion ensimmäisen version toiminnasta.

Toteutustapa vaati staattisen kokonaislukumuuttujan funktion sisällä pitämään muistissa viestin pituuden, mikäli viesti jäisi kesken. Yleisin syy viestin kesken jäämiselle oli UARTin hitaus. Mikro-ohjaimen UART-piiri ei ehtinyt kääntää sarjaliikenteellä vastaanotettuja merkkejä FIFO-rekisteriin siinä ajassa, jossa funktio ehti tarkastaa jo saapuneen merkin.

Lisäämällä viivytyksen funktion alkuun suurin osa viesteistä saatiin luettua yhdellä funktiokutsulla. UARTilla vastaanotettu viesti tallennettiin pääohjelmassa varattuun merkkijonopuskuriin (Kuvio 5).

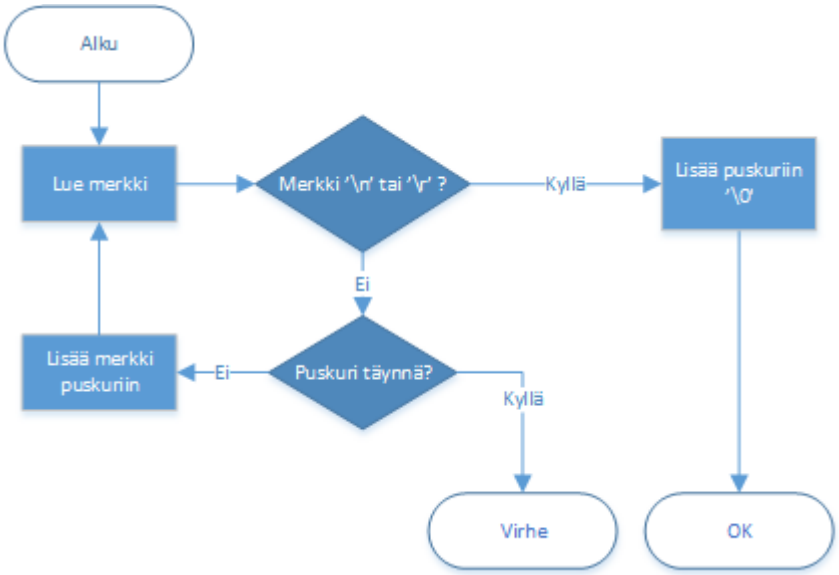


Kuvio 4. Tietovuokaavio UART-viestin vastaanottamisesta.

Lisäksi kirjastoon luotiin funktio `UART_getchar`, joka lukee yhden merkin FIFO-rekisteristä. Rekisterin ollessa tyhjä funktio jää odottamaan merkkiä tukkien samalla pääohjelman suorituksen. Funktio on käytännössä vain LibMF-kirjaston vastaavan funktion abstrahointi mukautetuilla palautusarvoilla.

5.4 Toinen versio UART-kirjastosta

UARTin toimintaa optimoitiin tarpeisiin sopivammaksi. Uudessa versiossa merkkejä odotetaan, kunnes tunnistetaan rivinvaihtomerkki (Kuvio 6). Tämä toteutus tukkii pääohjelman suorituksen UART-viestin luennan ajaksi. Tukinta ei haittaa, sillä UARTin toiminta kehitettävällä Sigfox-piirillä on määritelty siten, että UART-viestit tulee lähettää rivinvaihtomerkkiin päättyvinä merkkijonoina. UART-viestintään käytetyn PuTTY-ohjelman asetukset on mahdollista muokata määrittelyä vastaavaksi. Uusi versio `UART_readline`-funktiosta on aiempaa yksinkertaisempi ja toimii varmemmin. Funktion ohjelmakoodin rivien määrä lyheni ensimmäisen version 39:stä 14:ään.



Kuvio 5. Vuokaavio `UART_readline`-funktion toisesta versiosta.

Komentorajapinnan kehityksen ohessa kirjastoon lisättiin `UART_reply`-funktio, jota on tarkoitus käyttää viestien lähettämiseen. Funktiolle annetaan parametrina merkkijono, jonka funktio välittää UARTille ja sitä kautta vastaanottavalle osapuolelle. Käytettäessä kyseistä funktiota kaikkien lähtevien viestien lähettämiseen saadaan ohjelma-arkkitehtuuri pysymään modulaarisena.

5.5 Kehityksen aikaiset haasteet

Suurin haaste oli kehityksen aloitus. Ilman aiempaa kokemusta vastaavanlaisen kirjaston luomisesta oli vaikeaa keksiä, miten pääsisi alkuun. Myös mikro-ohjaimen arkkitehtuurin tuntemattomuus toi haastetta kehitykselle. Toiminnollisuutta päätettiin lähteä luomaan välittämättä liikaa ulkoasusta tai toteutustavasta, joita voitaisiin korjata jälkeenpäin. Tällä tavalla työhön päästiin käsiksi ja kehitys helpottui päivä päivältä mikro-ohjaimen ja kehitysympäristön tullessa tutummiksi. Ensimmäisen version karkean toiminnollisuuden saavuttamisen jälkeen tuotosta verrattiin vaatimusmäärittelyyn. Kun UARTin toimintamalli ja AX8052:n ohjelmointi oli jo tutumpaa, oli vaadittavan toiminnollisuuden luonnostelu paljon helpompaa ja toteutusta saatiin optimoitua merkittävästi.

5.6 Tulokset

Kehityksen tuotoksena syntyi kehitettävälle Sigfox-piirille sopiva UART-rajapinta. Rajapinta mahdollistaa viestin lähettämisen ja vastaanottamisen AX-SFEU-API:n ja työaseman tai toisen mikro-ohjaimen välillä. Viestin vastaanottamisen toiminnollisuus vastaa vaatimusmäärittelyä. Unitilasta heräämistä UARTin välityksellä ei ehditty tämän projektin puitteissa toteuttamaan.

Jatkokehityksenä UART-rajapinta kannattaa toteuttaa ilman LibMF:n apufunktioita. UART-keskeytyksiä tutkittiin hieman ja huomattiin, että LibMF:n UART-kirjasto käyttää UART0:n keskeytysvektoria, jolloin omaa keskeytyslogiikkaa ei pysty tekemään. Keskeytyksien käyttäminen mahdollistaisi erilaisia toteuttamistapoja rajapinnalle sekä unitilasta heräämisen UARTin kautta.

6 GPIO-RAJAPINNAN KEHITYS

6.1 Kehityksen tavoitteet

Tavoitteena on, että GPIO-portteja pystyttäisiin hyödyntämään mahdollisimman monipuolisesti. AX-SFEU-API-mikro-ohjaimen 10:llä GPIO-portilla on erilaiset käyttömahdollisuudet (Taulukko 1). Mahdollisista käyttötiloista ensisijaisen tärkeitä ovat portin ajo loogiseen 0- ja 1-tilaan (lähtö), portin tilan lukeminen (tulo), ylösvetovastuksen käyttö sekä keskeytykset. Analogisten signaalien tuloa (ADC) taikka lähtöä (DAC) ei toimeksiantajan puolesta tämän projektin puitteissa tarvita.

AX-SFEU-API:n GPIO-porttien toimintaan perehdytään mahdollisimman syvällisesti, minkä jälkeen porttien hallitsemiseksi luodaan muuta ohjelmointia helpottava kirjasto. GPIO-portteja hallinnoivien funktioiden tulisi olla mahdollisimman käyttäjäystävällisiä.

Taulukko 1. GPIO-porttien mahdolliset toimintatilat (ON Semiconductor 2016e, 4).

Portti	Lähtö ajo alas	Lähtö ajo ylös	Korkean impedanssin tulo	Tulo ylösvetovastuksella	ADC	DAC
GPIO0	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä
GPIO1	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä	Ei
GPIO2	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä	Ei
GPIO3	Kyllä	Kyllä	Kyllä	Kyllä	Kyllä	Ei
GPIO4	Kyllä	Kyllä	Kyllä	Kyllä	Ei	Kyllä
GPIO5	Kyllä	Kyllä	Kyllä	Kyllä	Ei	Ei
GPIO6	Kyllä	Kyllä	Kyllä	Kyllä	Ei	Ei
GPIO7	Kyllä	Kyllä	Kyllä	Kyllä	Ei	Ei
GPIO8	Kyllä	Kyllä	Kyllä	Kyllä	Ei	Ei

6.2 Sigfox-järjestelmäpiirin porttien kartoitus

Sigfox-piirin fyysiset portit kartoitettiin ennen ohjelmointityön aloittamista. Kytkentäkaaviosta selvitettiin, mille fyysisille nastoille GPIO-portit on kytketty. Esimerkkisovellusta

muokattiin niin, että mikro-ohjaimelle voitiin syöttää numero välillä 0–9 DebugLink-protokollalla. Ohjelma vaihtoi syötettyä numeroa vastaavan GPIO-portin tilan, jolloin jännitemittarilla pystyttiin tarkistamaan portin fyysinen sijainti piirillä.

Kartoituksessa huomattiin, että portteja 8 ja 9 ei pysty tuntemattomasta syystä ajamaan ylös eikä alas.

6.3 AX8052:n GPIO:n toiminta

AX8052F143-mikro-ohjaimessa jokaisella GPIO-portilla on oma 65 kΩ:n ylösvetovastus, joka voidaan kytkeä käyttöön ohjelmallisesti. Jokainen portti voidaan ohjelmoida keskeyttäväksi, jolloin keskeytys tapahtuu aina portin tilan muuttuessa. Portteja 0 – 3 voidaan käyttää analogisina tuloina. Mikro-ohjaimen herätessä syväunesta GPIO-portit tulisi kytkeä päälle kirjoittamalla 1 GPIOENABLE-rekisteriin. Kyseisen rekisterin arvon ollessa 0 kaikki GPIO-portit ovat ns. jäässä, jolloin niiden tila pysyy samana, vaikka niitä yritettäisiin ajaa tai tulosignaali muuttuisi. Asetettaessa mikro-ohjain unitilaan käyttäjän tulee varmistaa, että GPIO-portit eivät jää tilaan, jossa virrankulutus on liian suuri. (ON Semiconductor 2016d, 47–48.)

6.3.1 GPIO-rekisterit

AX8052-mikro-ohjaimessa GPIO-portit käsitetään oheislaitteina. GPIO-rekistereitä hallitaan XREG- ja SFR-osoiteavaruuksien kautta (ON Semiconductor 2016d, 15–16). SFR:ssä oheislaitteiden rekisterit sijaitsevat SFR:n ulkoisessa osoiteavaruudessa (ON Semiconductor 2016d, 10).

GPIO-portit on jaoteltu A-, B- ja C-rekisteriryhmiin. Lisäksi on olemassa R-rekisteri, joka on tarkoitettu radiopiirin käyttöön. Jokaisen rekisterin yksittäinen bitti vastaa yhtä ajettavaa porttia. AX8052F143-mikro-ohjaimessa on yhteensä 24 GPIO-porttia, joista AX-SFEU-API:ssa on GPIO:n käytössä 10 (Taulukko 2).

Taulukko 2. AX-SFEU-API:n GPIO-porttien sijainti rekistereissä (ON Semiconductor 2015, 41; ON Semiconductor 2016e, 4).

Bitti	0	1	2	3	4	5	6	7
A-rekisterit	GPIO0	GPIO1	GPIO2	-	-	GPIO3	-	-
B-rekisterit	-	-	-	GPIO9	-	-	-	-
C-rekisterit	GPIO4	GPIO5	GPIO6	GPIO7	GPIO8	-	-	-

Rekisteriryhmä sisältää yhteensä 6 (A:lla 7) rekisteriä, joiden avulla porttien tilaa hallitaan (Taulukko 3).

Taulukko 3. GPIO-rekisterit (ON Semiconductor 2015, 48–49).

Nimi	Luku/Kirjoitus	Kuvaus
PORT	Luku/Kirjoitus	Kun portti toimii tulona (DIR = 0), määrittää, käytetäänkö portissa ylösvetovastusta. Vastus käytössä = 1, ei käytössä = 0. Kun portti toimii lähtönä (DIR = 1), ajaa portin tilan ylös/alas. Ylös = 1, alas = 0.
PIN	Vain luku	Portin tilan lukemiseen, kun portti on tulo-tilassa (DIR = 0). Jos portti on asetettu keskeyttäväksi, PIN-rekisterin lukeminen nollaa keskeytyksen tilan PINCHG-rekisterissä.
DIR	Luku/Kirjoitus	Määrittää, toimiiko portti lähtö vai tulo-tilassa. Lähtö = 1, tulo = 0.
PALT	Luku/Kirjoitus	Kun portti on asetettu DIR-rekisterissä lähdöksi, määrittää, toimiiko portti ajettavassa lähtötilassa vai vaihtoehtoinen toimintatilassa.
INTCHG	Luku/Kirjoitus	Keskeytyksien hallinta. Keskeytys päällä = 1, pois päältä = 0.
PINCHG	Vain luku	Portin tilan vaihtumisen havaitsemiseen. Tila vaihtunut = 1, tila ennallaan = 0.

A-rekisteriryhmään kuuluu lisäksi ANALOG-rekisteri, jolla määritetään, toimiiko analogitulona. Porteilla on omia vaihtoehtoisia toimintatilat, joita voi kytkeä päälle PALT-rekisterissä (Taulukko 4).

Taulukko 4. GPIO-porttien vaihtoehtoiset toimintatilat (ON Semiconductor 2016d, 48).

Portti	Vaihtoehtoinen funktio			
GPIO0	T0OUT	IC1	ANALOG0	XTALP
GPIO1	T0CLK	OC1	ANALOG1	XTALN
GPIO2	OC0	U1RX	ANALOG2	COMPI00
GPIO3	IC0	U1TX	ANALOG5	COMPI10
GPIO4	SSEL	T0OUT	EXTIRQ0	-
GPIO5	SCK	T0CLK	COMPO1	-
GPIO6	SMOSI	U0TX	-	-
GPIO7	SMISO	U0RX	COMPO0	-
GPIO8	COMPO1	ADCTRIG	EXTIRQ1	-
GPIO9	OC0	T2CLK	EXTIRQ1	-

GPIO-porteilla on monia vaihtoehtoisia toimintatiloja. Vaihtoehtoisen toimintatilan kytkeminen päälle tarkoittaa sitä, että taulukossa 4 listatut toiminnot saavat portin käyttöönsä. Yleensä toiminnoille on saatavilla useampi portti, jolloin käytettävä portti määritellään toimintokohtaisissa rekistereissä. Käyttäjän vastuulle jää, ettei eri toiminnot käytä samaa porttia samanaikaisesti.

6.3.2 GPIO-porttien hallinta ohjelmallisesti

C-kieltä käytettäessä tarvittavat makrot GPIO:n hallinnalle löytyvät ax8052-kirjastosta. Makrojen nimet koostuvat rekisterien nimistä, joiden loppuun on lisätty rekisteriryhmän tunnus. Esimerkiksi A-rekisteriryhmän PORT-rekisteriä hallitaan `PORTA`-makron avulla. Jokaisessa rekisterissä jokainen bitti vastaa yhtä GPIO-porttia. Yksittäisiä GPIO-portteja voidaan hallita käyttämällä bittimaskeja.

6.4 Rajapinnan suunnittelu ja kehitys

GPIO-rajapintaa alettiin suunnitella On Semiconductorin esimerkkiohjelman päälle omana projektinaan. Kirjastolle päätettiin tehdä funktiot GPIO-portin toimintatavan asettamiselle, toimintatapoja vastaaville toiminnoille sekä GPIO-keskeytyksien lisäämisille. Valmiit funktiot muistuttavat ulkoisesti paljon Arduinon GPIO-funktioita, vaikkakin toimivat eri tavalla mikro-ohjainten arkkitehtuurillisien eroavaisuuksien vuoksi.

On Semiconductorin ohjeen mukaan portit, jotka eivät ole käytössä, tulisi asettaa tulotilaan ylösvetovastus kytkettynä. Tämä estää porttien jännitetason kellumisen, joka voi johtaa korotettuun virrankulutukseen. (ON Semiconductor 2016d, 48.) Porttien alustukselle luotiin `gpio_init`-funktio, joka alustaa kaikki portit tuloiksi ylösvetovastus kytkettynä. Funktiota tulisi kutsua mikro-ohjaimen käynnistyessä ennen flash-muistin kalibraatiodatan lataamista ja ennen GPIO:n kytkemistä päälle GPIOENABLE-rekisteriä muuttamalla (ON Semiconductor 2016d, 48).

GPIO-kirjaston funktioille suunniteltiin tapa, jolla funktion parametrina annettu GPIO-portti voidaan syöttää numerona välillä 0–9. GPIO-portteja vastaavista biteistä luotiin vakiomuuttujat (Ohjelmakoodi 2).

Ohjelmakoodi 2. GPIO-portteja vastaavat bitit.

```
/// GPIO Port magic numbers
#define GPIO0   0x01
#define GPIO1   0x02
#define GPIO2   0x04
#define GPIO3   0x20
#define GPIO4   0x01
#define GPIO5   0x02
#define GPIO6   0x04
#define GPIO7   0x08
#define GPIO8   0x10
#define GPIO9   0x08
```

Vakiomuuttujan ja parametrina annettavan numeron linkittämiseksi luotiin `gpio_map_t`-tietuetyyppi. Tietuetyypistä luotiin staattinen taulukko, johon sisällytettiin kaikkia GPIO-portit. Näin parametrina annetun portin numero `pin` saadaan helposti vastaamaan oikeaa bittiä halutussa rekisterissä (Ohjelmakoodi 3).

Ohjelmakoodi 3. Porttia vastaavan bitin määritys `pin`-parametrin mukaisesti.

```
uint8_t _bit = (gpio_map[pin].bit);
```

GPIO-portin tilaa muutettaessa tulee muistaa, että portit sijaitsevat eri rekisteriryhmissä. Rekisterin valintaan käytettiin switch-rakennetta (Ohjelmakoodi 4).

Ohjelmakoodi 4. Esimerkki switch-rakenteen käytöstä `GPIO_digitalWrite`-funktiosta. Bittioperaation `state`-muuttuja on funktion parametrina saatu int8_t-tyypin arvo, joka vastaa portin ajettavasta tilasta.

```
switch(pin) {
    case 0:
    case 1:
    case 2:
    case 3: PORTA ^= (-state ^ PORTA) & (_bit);
            break;

    case 4:
    case 5:
    case 6:
    case 7:
    case 8: PORTC ^= (-state ^ PORTC) & (_bit);
            break;

    case 9: PORTB ^= (-state ^ PORTB) & (_bit);
            break;

    default: ;
}
```

GPIO-kirjaston jokainen funktio toteutettiin samalla tyylillä, vaihtaen vain switch-rakenteen operaatioita.

Funktioiden toiminta testattiin ja varmistettiin esimerkkisovelluksen päälle luodulla sovelluksella sekä jännitemittarilla. Testatessa huomattiin, että portin toimintatilan ollessa tulo ja portin fyysisen nastan sekä ylösvetovastuksen ollessa kytkemättä portin jännite pääsee seilaamaan määräämättömästi. GPIO:n alustukseen liittyvä ohjeistus saatiin näin todennettua (ON Semiconductor 2016d, 48).

6.5 Keskeytyksen lisääminen ja poistaminen

8051-mikro-ohjainperheessä keskeytysvektoritaulukko alkaa aina muistin osoitteesta `0x03` (IAR Systems 2011, 75). AX8052F143-mikro-ohjaimella on yhteensä 22 keskey-

tysvektoria, joista 1 on GPIO-keskeytyksille ja 2 ulkoisille keskeytyksille (ON Semiconductor 2016d, 11). Keskeytysfunktio ilmoitetaan IAR:n kääntäjällä käyttämällä `__interrupt`-avainsanaa sekä `pragma`-direktiiviä yhdessä keskeytysvektorin kanssa. Keskeytysfunktioiden ei tule palauttaa mitään. (IAR Systems 2011, 75.)

GPIO-kirjastoon luotiin funktiot GPIO-keskeytyksen lisäämiselle ja poistamiselle. Keskeytyksen lisäämiseksi luodulle `GPIO_attachInterrupt`-funktiolle annetaan parametrina käytettävä portti sekä osoitin käyttäjän luomaan keskeytysfunktioon. Käyttäjän luoman funktion osoitin tallennetaan GPIO-moduulissa varattuun funktio-osoittimeen. GPIO-moduuliin luotiin keskeytysfunktio, jossa käyttäjän luomaa funktiota kutsutaan paikallisen funktio-osoittimen avulla (Ohjelmakoodi 5). Käytettävä portti asetetaan keskeyttäväksi aiemmin mainitulla switch-rakenteella muuttaen porttia vastaava bitti 1:ksi INTCHG-rekisterissä. Lisäksi `GPIO_attachInterrupt`-funktiossa kytketään GPIO-keskeytykset päälle. Vastaavasti keskeytyksen poistamiseen luotiin `GPIO_detachInterrupt`-funktio, joka nollaa porttia vastaavan bitin INTCHG-rekisterissä. Jos käyttäjä haluaa käyttää useampaa GPIO-porttia keskeyttävänä, tulee samaan keskeytysfunktioon lisätä käsittely kaikille keskeyttäville porteille. Portin tilan muuttumisen voi tarkistaa PINCHG-rekisteristä.

Ohjelmakoodi 5. GPIO-moduulin paikallisen osoittimen käyttäminen käyttäjän keskeytysfunktion kutsumiseen.

```
void (*gpio_isr)();

#pragma vector=0x001B
__interrupt void gpio_interrupt()
{
    (*gpio_isr)();
}
```

AX8052F143-mikro-ohjaimella on myös kaksi keskeytysvektoria ulkoisille keskeytyksille, EXTIRQ0 ja EXTIRQ1. Ulkoisia keskeytyksien käyttöä hallitaan omilla MODE- ja PIN-rekistereillä. PIN-rekistereillä (EXTIRQ0PIN ja EXTIRQ1PIN) asetetaan keskeyttävä GPIO-portti. EXTIRQ0:n porttivaihtoehdot ovat GPIO-portit PB0 tai PC0. EXTIRQ1 käyttää PB3- tai PC4-porttia. AX-SFEU-API:n tapauksessa EXTIRQ0:aa on mahdollista käyttää vain PC0-portilla, sillä PB0 on varattu mikro-ohjaimen suorittimen aktiivisuusledin käyttöön (ON Semiconductor 2016e, 4). Ulkoiset keskeytykset on mahdollista asettaa keskeyttämään signaalin tilan ollessa ylhäällä tai alhaalla, nousevasta reunasta tai laskevasta reunasta muuttamalla MODE-rekisterin arvoa. (ON Semiconductor 2016d, 49.)

Keskeytyksille on mahdollista asettaa kaksi eri prioriteettia: matala tai korkea. Korkean prioriteetin keskeytykset voivat keskeyttää matalan prioriteetin keskeytyksien käsittelyn. Korkean prioriteetin keskeytyksen käsittelyä ei voi keskeyttää. Jokaisen keskeytysvektorin prioriteettia voidaan hallita SFR-osoiteavaruudessa sijaitsevalla bitillä. Oletusprioriteetti on aina matala. (ON Semiconductor 2016d, 11.)

6.6 Kehityksen tulokset

Kehityksen tuotoksena syntyi GPIO-kirjasto, joka sisältää tarvittavat funktiot GPIO-portin toimintamuodon asettamiseen, ajamiseen lähtönä, tulosignaalin lukemiseen sekä keskeytyksen lisäämiseen ja poistamiseen. Funktiot toteutettiin muistuttamaan Arduinon GPIO-funktioita. Toteutetut funktiot auttavat erilaisten sovelluksien ohjelmoinnissa, tehden järjestelmäpiirin käytöstä helpompaa etenkin ohjelmoijille, joille AX8052-mikro-ohjain ei ole tuttu.

Jatkokehityksenä GPIO-rajapintaan olisi hyvä lisätä toiminnot analogisten signaalien lukemiseen ja ajamiseen. Keskeytyksille olisi hyvä luoda toiminnollisuus, jolla käyttäjä voi valita keskeytystavaksi GPIO-keskeytyksen tai ulkoisen keskeytyksen. Keskeytyksen prioriteettien hallintaan olisi myös syytä kehittää käyttäjäystävällinen funktio.

GPIO-rajapinnan kehitys oli opinnäytetyön projektityön osana erittäin sopiva ja mielenkiintoinen. Kehitys oli suhteellisen helppo aloittaa, kun aiempaa kokemusta löytyi muiden mikro-ohjaimien GPIO-toteutuksien käytöstä. Rajapintaa lähdettiin kehittämään mallilla, joka oli itselle tuttu. Epäselväksi kehityksen aikana jäi, miten optimaalinen GPIO-porttien tilojen käsittely on rajapinnassa käytetyillä algoritmeilla. Jatkokehityksessä voitaisiin siis myös tutkia toteutettujen algoritmien tehokkuutta ja verrata niitä vaihtoehtoisiin toteutustapoihin.

7 KOMENTORAJAPINNAN KEHITYS

7.1 Vaatimusmäärittely ja tavoitteet

Komentorajapinnan tulisi sisältää UART-protokollan kautta lähetettyjen viestien tulkinnan ja viestien sisältämien komentojen suorituksen. Käytettävät komennot voidaan määritellä kehityksen yhteydessä. Vaadittuja komentoja ovat järjestelmäpiirin asetuksien lukeminen ja muokkaaminen, Sigfox-tunnisteiden, lämpötilan ja käyttöjännitteen lukeminen, Sigfox-sertifiointiin tarvittavien testien ajaminen sekä Sigfox-viestien lähettäminen ja vastaanottaminen.

Tavoitteena projektin komentorajapinnan kehityksessä oli luoda vaatimusmäärittelyn mukainen toiminnollisuus. Kehityksessä otettiin käyttöön testivetoisen kehityksen kehitysmalli, jonka hyötyjä ja haittoja havainnoitiin kehityksen aikana. Alustava tieto testivetoisesta kehityksestä oli kerätty UART- ja GPIO-rajapintojen kehityksien ohessa.

7.2 Komentorajapinnan ensimmäinen versio

Komentorajapinnan kehitykseen otettiin mukaan testivetoisen kehityksen kehitysmalli. Testit luotiin tavallisena komentoriviohjelmana hyödyntämättä mitään valmista yksikkötestikehystä. Komentorajapinnan käyttämien kirjastojen otsikkotiedostot kopioitiin testiympäristöön, ja niiden sisältämistä funktioista tehtiin tynkäfunktiot omaan lähdekooditiedostoon. Tynkäfunktioilla tarkoitetaan funktiota, joka korvaa otsikkotiedoston alkuperäisen funktion ja sisältää ennalta määrätyn arvon palautuksen.

7.2.1 Komentosyntaksin määritys

Komentojen tyylin määrityksessä tutkittiin kaupallisen piirin komentojen tyyliä. AX-SFEU toimii UART-protokollalla käyttäen AT-komentoja. AX-SFEU:ssa kaikki komennot alkavat ”AT”-merkkijonolla, jonka jälkeen komento jatkuu itse komennolla ja mahdollisilla parametreilla. Komento loppuu tyhjätilamerkkiin, eli välilyöntimerkkiin, sarkainmerkkiin, rivinvaihtomerkkiin tai vastaavaan. Komennon suorituksen onnistuessa piiri vastaa ”OK”-merkkijonolla. Jos komento palauttaa jonkin arvon, vastaus sisältää vain arvon lukeman.

Komento ”AT” on yksinkertainen komento yhteyden testaamiseen, ja se palauttaa aina merkkijonon ”OK”. (ON Semiconductor 2016e, 11-12.)

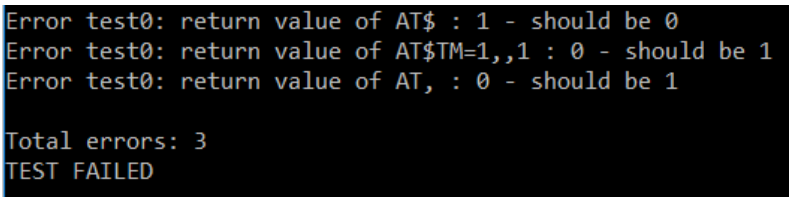
Komentorajapinnan komentosyntaksi määriteltiin vastaamaan AX-SFEU:n komentoja. Aluksi komentorajapintaan lisättiin komennot vain lämpötilan ja tunnisteiden lukemiselle sekä testien ajamiselle (Taulukko 5).

Taulukko 5. Komentorajapinnan alustavasti määritellyt komennot.

Komento	Kuvaus
AT	Yksinkertainen komento yhteyden testaamiseen.
AT$T?	Lämpötilan luku
AT$ID	ID-tunnisteen luku
AT$PAC	PAC-tunnisteen luku
AT$TM=	Testin ajo, parametreina annetaan testin numero ja testikohtainen parametri pilkulla erotettuna

7.2.2 Testit

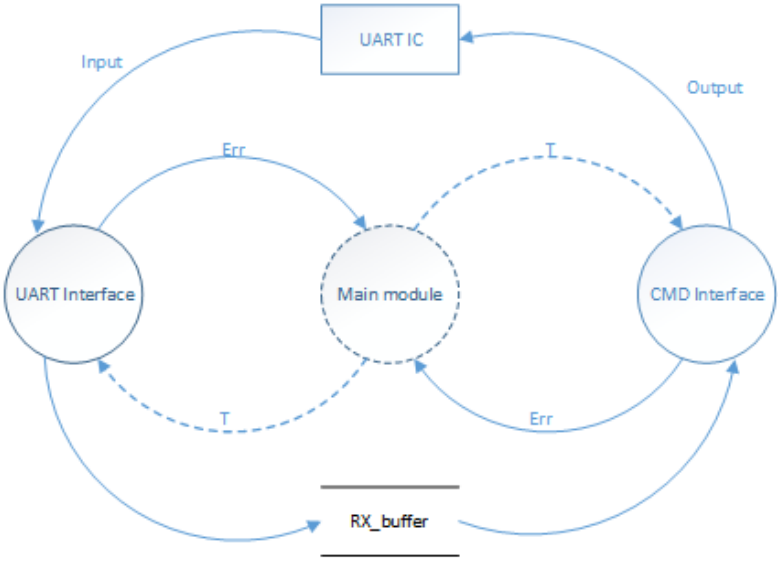
Ennen rajapinnan kehittämisen aloittamista luotiin testejä, joissa komentorajapinnalle syötettiin erilaisia komentoja. Syötettävien komentojen joukko sisälsi oikeanlaisia komentoja sekä komentoja, jotka sisälsivät kirjoitusvirheitä tai virheellisiä parametreja. Testiohjelma ohjelmoitiin tulostamaan virheviesti jokaisesta epäonnistuneesta testistä (Kuva 11). Testivetoisen kehityksen mukaisesti rajapintaa alettiin kehittää niin, että kaikki testit onnistuisivat.



Kuva 11. Kuvankaappaus testien tuloksista.

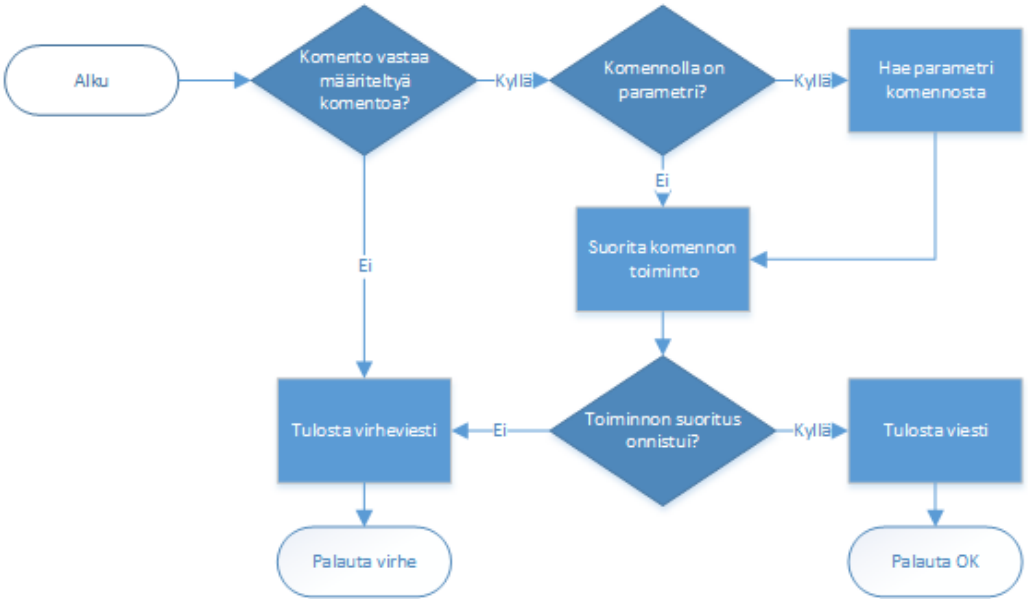
7.2.3 Rajapinnan kehitys

Rajapinnalle luotiin alustavasti yksi funktio, joka näkyy pääohjelmalle. Funktio sai pääohjelmalta parametrina UARTin kautta vastaanotetun komennon muistiosoitteen, jonka sisältämä komento tulkittiin ja komennon mukainen toiminto suoritettiin. Komentomoduuli tulosti palautusviestit käyttäen LibMF:n UART-kirjastoa. Komentorajapinnan toiminta yhdessä pääohjelman ja UART-rajapinnan kanssa on kuvattu kuviossa 7.



Kuvio 6. Tietovuokaavio komentojen lähettämisestä ja vastaanottamisesta komentorajapinnan ensimmäisen version kanssa.

Komentojen tulkinnassa parametrina saatua komentoa verrattiin jokaiseen mahdolliseen komentovaihtoehtoon. Kaikki määritellyt komennot oli varattu komentomoduulissa omaan taulukkoonsa. Jokaisen komennon määritelmä sisälsi ”AT$”-etuliitteen. Testien vaatima parametri tulkittiin erikseen silloin, kun ”AT$TM=”-merkkijono oli tunnistettu. Komentoja vastaavat toiminnot suoritettiin, minkä jälkeen pääohjelmalle palautettiin kokonaislukumuuttuja merkiksi komennon suorittamisen onnistumisesta tai epäonnistumisesta. Ensimmäisen version toiminta on kuvattu kuviossa 8.



Kuvio 7. Vuokaavio komentorajapinnan ensimmäisen version toiminnasta.

Ensimmäisellä versiolla pystyttiin lukemaan Sigfox-tunnisteita ja ajamaan testejä. Rajapinnan toteutuksessa ilmeni kuitenkin useita virheitä, ja toteutustavasta johtuen lisäominaisuuksien lisääminen todettiin hankalaksi. Toimeksiantajayrityksen yhteyshenkilöltä saatiin arvokkaita ohjeita rajapinnan suunnitteluun, ja niiden pohjalta lähdettiin toteuttamaan uutta ratkaisua.

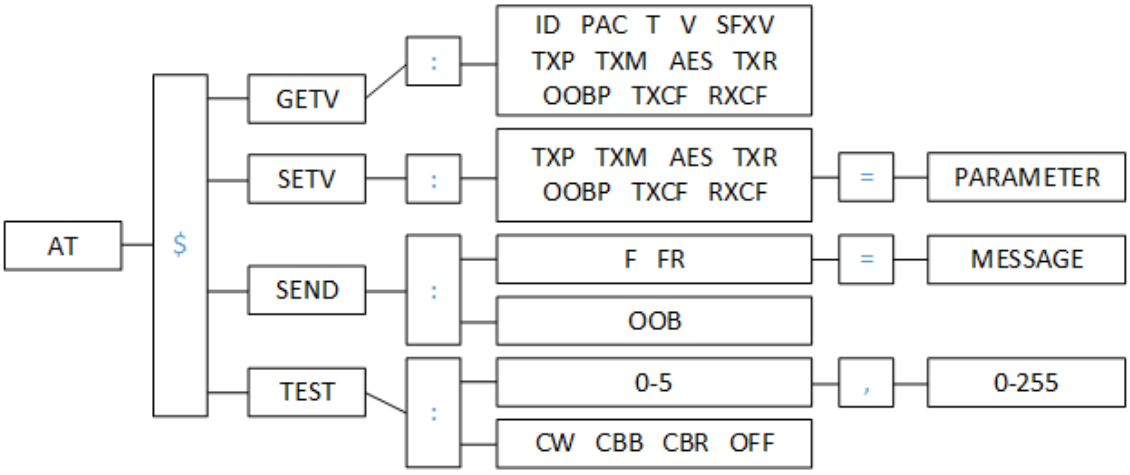
7.3 Komentorajapinnan toinen versio

Saadun palautteen perusteella komentorajapinnan toteutustapaa päätettiin muuttaa. Komentojen syntaksia muokattiin vastaamaan paremmin tarpeita ja komentojen tulkintaan luotiin uudet algoritmit.

7.3.1 Komentosyntaksin uudelleenmäärittely

Uuden komentosyntaksin suunnittelussa otettiin huomioon myös toteutettavuus. Komennot suunniteltiin sisältämään etuliite, pääkomento, alikomento sekä alikomennon parametri (Kuvio 9). Etuliite ”AT” sekä etuliitteen ja pääkomennon välinen erottelumerkki ’$’ päätettiin jättää ennalleen. Komennon loppuosa määriteltiin kokonaan uudelleen. Pääkomennot määriteltiin asetuksien ja tunnisteiden asettamiselle sekä lukemiselle, testeille sekä viestien lähetykselle. Pääkomentojen pituudet määriteltiin yhtäläisiksi ohjelmoinnin

helpottamiseksi. Arvojen lukemiseen ja asettamiseen luotiin alikomennot kaikille arvoille, joita pystyy On Semiconductorin ohjelmointirajapinnan avulla muokkaamaan. Testien alikomennot määriteltiin rakentumaan testiä vastaavasta kokonaisluvusta välillä 0–5 sekä parametria vastaavasta, yhden tavun kokoisesta kokonaisluvusta välillä 0–255. Testin numero ja parametri erotellaan pilkulla. Viestin lähetykseen määriteltiin alikomennot viestin lähetykselle ilman vastausta, viestin lähetykselle vastauksella sekä OOB-viestin lähetykselle.

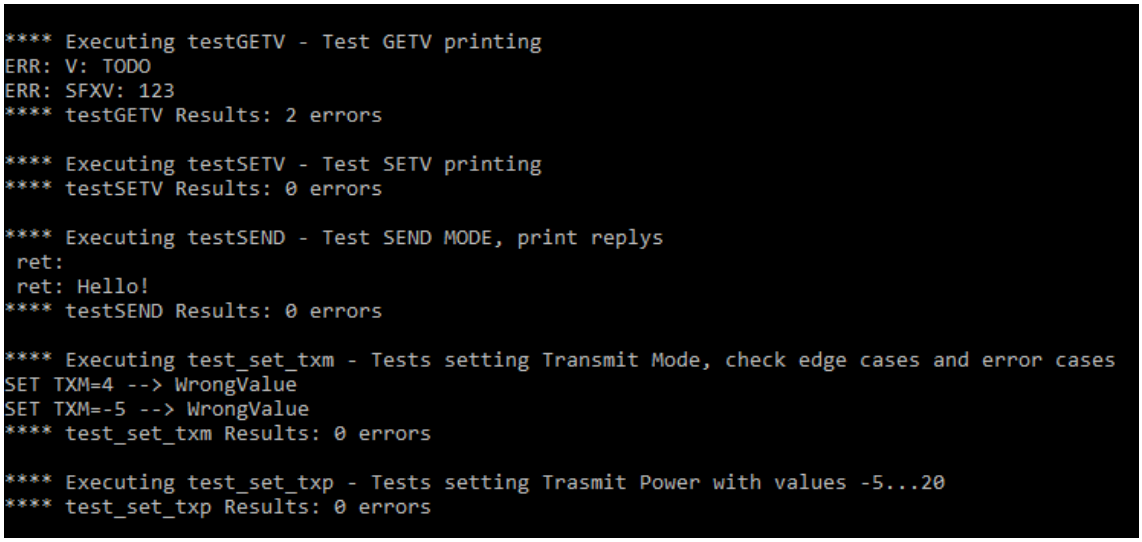


Kuvio 8. Puukaavio uudelleen määriteltyjen komentojen rakentumisesta.

Pääkomennon ja alikomennon erotteluun määriteltiin käytettäväksi kaksoispistettä. Testin ajamiseen tarvittavien parametrien erotteluun alikomennosta määriteltiin käytettäväksi pilkkua. Viestin lähetyksessä lähetettävä viesti annetaan alikomennon jälkeen ’=’-merkillä eroteltuna. Oikeanlainen komento ”HelloWorld”-viestin lähettämiseksi vastausikkunalla olisi ”AT$SEND:FR=HelloWorld\n”. Arvojen asettamisessa alikomennon ja parametrin erotteluun käytetään ’=’-merkkiä.

7.3.2 Testit

Komentorajapinnan toisen version kehitys tapahtui pääosin testiympäristössä (Kuva 12). Kun rajapinta toimi testiympäristössä halutulla tavalla ilman suurempia vikoja, siirryttiin mikro-ohjaimen kehitysympäristön puolelle, jossa funktiot ja muuttujat hienosäädettiin mikro-ohjaimelle sopivaksi.



Kuva 12. Kuvankaappaus komentorajapinnan yksikkötestien tuloksista.

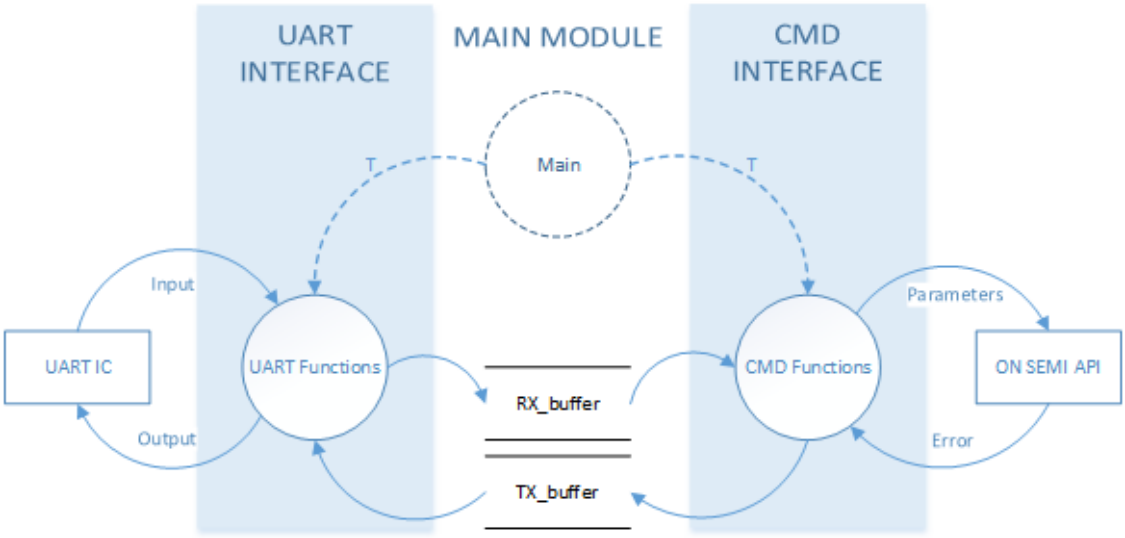
Toteutettu rajapinta syntyi pitkälti testien mukaisesti. Aluksi luotiin testi etuliitteen tunnistukselle. Kun testi meni läpi, siirryttiin pääkomennon tunnistukseen. Tällä tavalla edettiin, kunnes rajapinta oli muodostunut. Rajapinnan toteutustapa muuttui kehityksen aikana moneen otteeseen, mutta vanhat testit mahdollistivat sen, että varmuus ohjelmakoodin osien toiminnasta säilyi.

Komentorajapinnan kehityksen aikana kehitettiin myös testiympäristöä luomalla kevyt ohjelmakehys yksikkötesteille. Kehyksen luominen oli hyödyllinen, sillä se helpotti uusien testien kirjoittamista sekä testien tuloksien lukemista. Oman kustomoidun testikehyksen käyttäminen ei ole kuitenkaan pidemmän päälle suositeltavaa, sillä suurempien projektien kanssa kehyksen pitäminen yhtenäisenä on haastavaa. Kustomoidun testikehyksen ylläpitäminen vaatii enemmän työtä kuin valmiin ja toimivaksi todetun kehyksen käyttäminen. (Grenning 2011, 94.)

7.3.3 Toisen version kehitys

Komentorajapinnan toinen versio suunniteltiin toimimaan siten, ettei komentorajapinnan sisällä tapahtuisi UART-tulostuksia. Pääohjelmamoduulissa luotiin `rx_msg`-merkkijonopuskurin lisäksi merkkijonopuskuri `tx_msg`, johon tallennettaisiin lähetettävät viestit. Molempien puskurien osoittimet annetaan rajapinnan pääohjelmalle näkyvälle `CMD_run`-funktiolle parametreina yhdessä `tx_msg`-puskurin pituuden kanssa. Komentorajapinnan

sisällä komento käsitellään osa kerrallaan. Lopuksi komennon mukainen toiminto suoritetaan. Palautettava viesti tallennetaan `tx_msg`-puskuriin. Uudella toteutustavalla UART-tulostukset saatiin pois komentorajapinnasta ja sitä myötä rajapinnasta saatiin modulaarisempi. Alla on esitetty komentorajapinnan uuden version toiminta yhdessä pääohjelman ja UART-rajapinnan kanssa (Kuvio 10).



Kuvio 9. Tietovuokaavio komentorajapinnan toisen version toiminnasta yhdessä pääohjelman ja UART-rajapinnan kanssa.

Komentojen tulkinta

Komentorajapinnalle luotiin `CMD_run`-funktion lisäksi 5 funktiota komentojen tulkitsemiseen. Etuliitteen tunnistamisen jälkeen tunnistetaan pääkomento. Jokaiselle pääkomennolle on oma käsittelyfunktio, jonne siirrytään pääkomennon tunnistamisen jälkeen. Pääkomennon käsittelyfunktiossa suoritetaan komentoa vastaava toiminto. Palautettava viesti kulkee samaa polkua takaisin `CMD_run`-funktiolle, josta se palautetaan pääohjelmalle.

Pääkomennot sekä asetuksien lukemisen ja asettamisen komennot toteutettiin luomalla tietuetyyppi `cmd_t`. Tietuetyyppi sisältää vakiomuuttujat komentoa vastaavalle merkkijonolle sekä kokonaislukutyypin tunnistemuuttujalle (Ohjelmakoodi 6).

Ohjelmakoodi 6. Tietuetyyppi `cmd_t`.

```
static struct cmd_t{
    const char __code *cmd;
    const uint8_t id;
};
```

Tietuetyypistä luotiin taulukko, johon alustettiin kaikki tarvittavat komennot (Ohjelmakoodi 7).

Ohjelmakoodi 7. Asetuksien asettamiselle tarkoitetut komennot alustettuna `cmd_t`-tietuetyypin taulukossa.

```
const struct cmd_t __code setv_cmds[] = {
    {"TXP", SET_TXP},
    {"TXM", SET_TXM},
    {"AES", SET_AES},
    {"TXR", SET_TXR},
    {"OOBP", SET_OOBP},
    {"TXCF", SET_TXCF},
    {"RXCF", SET_RXCF}
};
```

Komennon tunnistus tapahtuu komennon osaa vastaavan funktion alussa. `Cmd_t`-tietuetyypistä luodaan osoitin, joka asetetaan osoittamaan haluttuun komentotaulukkoon. Silmukassa verrataan käyttäjän syötettä taulukon komentoihin (Ohjelmakoodi 8).

Ohjelmakoodi 8. Komennon tunnistukseen käytetty algoritmi.

```
uint8_t id = CMD_NONE;
uint8_t x = 0;
const struct cmd_t __code *pCmd = setv_cmds;

for(x = 0; x < get_cmd_count(setv_cmds); x++, pCmd++){
    if(strncmp_P(set_cmd, pCmd->cmd, strlen_P(pCmd->cmd)) == 0){
        id = pCmd->id;
        break;
    }
}
```

Yhtäläisten merkkijonojen löytyessä funktion paikalliseen tunnistemuuttujaan asetetaan komentoa vastaava tunniste, jonka avulla oikea komento suoritetaan switch-rakenteessa tarpeellisten tarkistuksien jälkeen.

Merkkijonon palautus

Komentorajapinta palauttaa aina merkkijonon, joka voidaan tulostaa UART-rajapinnan `UART_reply`-funktiolla. On Semiconductorin ohjelmointirajapinta sen sijaan palauttaa erityyppisiä arvoja. Standardikirjastoissa ei ollut AX8052-mikro-ohjaimelle sopivia funktioita eri arvojen muuttamiseksi merkkijonoiksi, joten sellaiset luotiin itse. Palautettavat arvot olivat uint8_t-, uint16_t-, uint32_t- sekä float-tyyppisiä. Arvoja tuli pystyä palauttamaan 10- ja 16-kantaisina.

Etumerkittömien kokonaislukujen muuntamiseen merkkijonoksi luotiin `u32toa`-funktio. Funktio on johdettu K&R:n `itoa`-funktiosta (Kernighan & Ritchie, 64). Funktioon on lisätty parametrit halutulle kantaluvulle sekä merkitsevien lukujen vähimmäismäärälle. Liukulukujen tulostukseen luotiin `ftoa`-funktio, jonka toteutuksessa käytettiin apuna `u32toa`-funktiota. Funktiolla voidaan tulostaa float-tyypin arvoja halutulla desimaalitarkkuudella. Ajan puutteen vuoksi `ftoa`-funktiolle ei ehditty luomaan logiikkaa lukujen pyöristämiselle.

Virheviestien palautukseen luotiin kehitystä helpottava makro, jota kutsutaan tarkistuksien yhteydessä (Ohjelmakoodi 9).

Ohjelmakoodi 9. Makro virheviestin palautukselle.

```
#define _return(buf, msg)    do{memcpy_P(buf, msg, sizeof(msg)); return buf;}while(0)
```

Makrolle annetaan parametrina `tx_msg`-merkkijonopuskurin osoite sekä itse viesti.

Käyttöjännitteen lukeminen

Käyttöjännitteen lukemiselle ei ollut valmista funktiota LibMF-kirjastoissa eikä On Semiconductorin ohjelmointirajapinnassa. Käyttöjännitteen lukeminen toteutettiin hallitsemalla ADC-rekistereitä (Ohjelmakoodi 10).

Ohjelmakoodi 10. Käyttöjännitteen lukeminen.

```
case GET_V:     ADCCH2CONFIG = 0xD9; /** Magic num for VDDIO */
                ADCCLKSRC = 0x00; /** Use FRCOSC */
                EA = 0;
                    ADCCONV = 0x01; /** Start conversion */
                    while((ADCCONV & 0x40) || (ADCCONV & 0x20) || (ADCCONV & 0x01)) ;
                    i = ADCCH2VAL1;
                    i = (i << 8) | ADCCH2VAL0;
                EA = 1;
                ADCCH2CONFIG = 0xFF; /** Turn CH2 Off */
                v = (float)i * 10.0f/65536.0f - 4.5f;
                return ftoa(v, ret, 3);
```

ADC-kanavan CONFIG-rekisterissä voidaan valita toiminnoksi käyttöjännitteen mittaus. ADC:lle asetetaan käytettävä kellolähde ADCCLKSRC-rekisterissä. ADC:ta voidaan ajaa ajastimella, mikäli ohjelman suoritusta ei haluta tukkia. (ON Semiconductor 2016d, 53.) Komentorajapinnan osalta tukinta ei haittaa, joten muunnos ohjelmoitiin tapahtumaan kertaheitolla. Yllä esitetyssä ohjelmakoodissa kellolähteenä käytetään nopeaa FRCOSC-oskillaattoria, jolloin muunnokseen kuluu mahdollisimman vähän aikaa. Käyttöjännitteen selvittämiseksi käytetään On Semiconductorin tarjoamaa kaavaa

$$VDDIO = ADCCHxVAL * \frac{10\ \mathrm{V}}{2^{16}} - 4{,}5\ \mathrm{V}.$$

Kaava 1. Käyttöjännitteen laskeminen muunnostuloksesta (ON Semiconductor 2016d, 55).

ADC:n muunnostulos tallentuu 16-bittisiin CHxVAL0- ja CHxVAL1-rekistereihin, joissa ’x’ vastaa käytettävän kanavan numeroa. Rekistereiden arvot yhdistetään yhteen uint32_t-tyypin muuttujaan. Käytetty kanava kytketään pois päältä ja kaavan mukainen laskutoimitus suoritetaan. Saatu käyttöjännitettä vastaava liukulukumuuttuja muutetaan merkkijonoksi `ftoa`-funktiolla.

7.4 Kehityksen aikaiset haasteet

Komentorajapinnan kehityksen aikana kohdattiin haasteita, jotka hidastivat kehitystyötä. Haasteet hidastivat merkittävästi kehitystä, ja siksi myös aikataulu venyi. Toimeksiantajan tuoteprototyypin toiminnollisuuden toistamista uudella piirillä ei ehditty kehittämään ajan puutteen vuoksi. Kohdatut haasteet saatiin kuitenkin ratkaistua.

7.4.1 Alustojen väliset erot

Testiympäristössä luodun toiminnollisuuden tuominen mikro-ohjaimelle oli haastavaa. Toteutettaessa rajapintaa testiympäristössä täyttä ymmärrystä mikro-ohjaimelle saatavilla olevista kirjastoista ei ollut. Esimerkiksi merkkijonon palautus toteutettiin aluksi testiympäristössä käyttämällä `snprintf`-funktiota. IAR:n 8051-kääntäjälle on saatavilla `sprintf`-funktio, jota käyttämällä ohjelmakoodi kääntyi, mutta ei toiminut. Syytä kyseisen funktion toimimattomuudelle ei saatu selville. Funktion käytöstä oli helppo luopua, sillä se käyttää alustasta riippumatta paljon muistia (Jones 2009).

7.4.2 Ongelma järjestelmäpiirin kanssa

Suurin haaste oli kehityksen aikana ilmennyt ongelma järjestelmäpiirin ja ohjelmointiympäristön kanssa. Ongelma havaittiin, kun järjestelmäpiiri ei alkanut toimia ohjelman latauksen jälkeen, vaikka ohjelmakoodi ja muut asetukset olivat pysyneet muuttumattomina. Järjestelmäpiiri ei vastannut UART-viesteihin. Kun mikro-ohjain pysäytettiin virheenjäljittäjällä, ohjain näytti olevan koko ajan valmiustilassa. Vikaa tutkittiin ja vian aiheuttajan jäljille päästiin, kun ohjelmakoodia muutettiin siten, että mikro-ohjaimen käynnistyessä ei ladattu Sigfox-tehdaskalibraatiotietoja flash-muistista. Tällä tavoin mikro-ohjain suoritti luotua ohjelmakoodia normaalisti, mutta Sigfox-asetuksien puuttuessa kaikki radiotaajuusasetukset olivat väärin, Sigfox-tunnisteita ei pystynyt lukemaan eikä Sigfox-viestintä ollut mahdollista. Kehitetyt rajapinnat toimivat normaalisti. Vika saatiin kohdennettua ongelmaan flash-muistin kanssa.

Vianetsintään otettiin mukaan AXSDB-komentoriviohjelma. AXSDB:llä on mahdollista hallita mikro-ohjaimen muistia sekä tallentaa ja ladata kalibraatiotietoja. AXSDB:llä mikro-ohjainta pystytään tutkimaan tarkemmin kuin AxCodeBlocks-ohjelmointiympäristöllä. Graafisen näkymän puute tekee AXSDB:n käytöstä vaikeampaa silloin, jos tarvitsee seurata ajettavan ohjelmakoodin suoritusta.

AX-SFEU-API:n kalibraatiotiedot sijaitsee flash-muistin viimeisen 1 kt:n alueella osoitteiden 0xFC00–0xFFFF välillä. Lukemalla kalibraatiotietojen muistilohko AXSDB:llä huomattiin, että kalibraatiotiedot olivat nollautuneet. Kalibraatiotiedot luettiin toiselta, käyttämättömältä piiriltä, josta pystyttiin selvittämään mm. ID- ja PAC-tunnisteiden sijainti. Ka-

libraatiotiedot tallennettiin toimivalta piiriltä ja ladattiin nollautuneelle piirille. Tämän jälkeen nollautunut piiri alkoi toimia normaalisti käyttäen samoja kalibraatiotietoja toisen piirin kanssa. Aiemmin nollautuneen piirin tunnistetiedot oli aiemmin kirjoitettu talteen, joten vanhojen tunnisteiden kirjoittamista suoraan toisen piirin kalibraatiotietoihin yritettiin. Toimenpide ei toiminut nollautuneen piirin elvyttämisessä. Kalibraatiotiedoissa on luultavasti jonkinlainen tarkistus tunnisteille, jota ei tämän projektin osalta selvitetty.

Vikaan johtanutta ongelmaa ei saatu selvitettyä, mutta se saatiin kierrettyä tallentamalla jokaisen kehitettävän piirin kalibraatiotiedot erilliseen tiedostoon työasemalle ennen ohjelmakoodin lataamista piirille. Jos tiedot tuntemattomasta syystä nollautuvat, voidaan tiedot ladata piirille uudestaan. AXSDB:tä käytettäessä huomattiin myös, että komentoriviohjelmalla ohjelmakoodin lataus toimii varmemmin kuin AxCodeBlocks-ympäristössä. AXSDB:llä kalibraatiotiedot eivät nollautuneet kertaakaan, kun taas AxCodeBlocks-ympäristössä ongelmaan törmättiin useaan otteeseen.

7.5 Tulokset

Kehityksen tuotoksena syntyi rajapinta, joka tulkitsee sille syötettyjä komentoja ja suorittaa komennon mukaisen toiminnon. Komentojen rakenne määriteltiin vastaamaan tarpeita. Rajapinta toteutettiin niin, että komentojen lisäys on jatkossa suhteellisen helppoa. Rajapinta suunniteltiin toimimaan pääsääntöisesti kehitetyn UART-rajapinnan kanssa, mutta toteutustavan ansiosta molempia rajapintoja voidaan käyttää toisistaan riippumatta. Kehityksen aikana tavattiin haasteita, jotka pitkittivät rajapinnan kehitystä, ja siksi alkuperäisestä aikataulusta ei pystytty pitämään kiinni. Kehityksen aikaiset ongelmat onnistuttiin selvittämään.

Komentorajapinnan kehityksessä sovellettiin testivetoista kehitystä. TDD:n todettiin olevan tehokas tapa ohjelmiston kehityksessä, mutta se toi mukanaan myös ongelmia. Suurin ongelma oli ohjelmakokonaisuuden siirtäminen testiympäristöstä mikro-ohjaimelle. Mikro-ohjaimeen ja kääntäjään tutustuttaessa tarkemmin kyseinen ongelma voidaan kuitenkin välttää. Vaikka TDD:tä käytettiin hieman sovellettuna ja ensimmäistä kertaa, pystyttiin sen tuomia etuja hyödyntämään. Välitön palaute kehitettyjen komponenttien toiminnasta osoittautui erittäin hyödylliseksi. Vanhat testitapaukset varmistivat, että uudet toiminnollisuudet eivät rikkoneet ohjelmakokonaisuutta. Testivetoinen kehitys toi varmuuden kehitetyn rajapinnan oikeanlaisesta toiminnasta.

8 YHTEENVETO

Opinnäytetyössä tutkittiin esineiden internetin laitteille kehitetyn Sigfox-verkon toimintaa sekä testivetoista kehitystä. Projektityön tavoitteina oli kehitettää Sigfox-järjestelmäpiirille rajapinnat, jotka mahdollistavat piirin käytön asiakkaille räätälöidyissä ratkaisuissa. Vaadittuja rajapintoja olivat UART- ja GPIO-rajapinnat sekä rajapinta UART-viestien käsittelyyn. Tavoitteena oli myös käyttää testivetoista kehitystä rajapintojen kehityksessä.

Rajapinnoista jäi puuttumaan joitakin ominaisuuksia, joita ei ehditty ajan puutteen vuoksi kehittää. Toimeksiantajan kannalta tärkeimmät tavoitteet kuitenkin saavutettiin ja toimeksiantajayritys oli tyytyväinen tuloksiin. Rajapinnat kehitettiin osittain testivetoisen kehityksen periaatteita soveltamalla. Projektityön tuotoksina onnistuttiin luomaan rajapinnat, jotka pääosin täyttävät niille määritellyt vaatimukset.

Projektityö aloitettiin UART-ja GPIO-rajapintojen kehityksellä. Rajapintojen kehityksen aikana opittiin AX8052-mikro-ohjaimen arkkitehtuurista ja toiminnasta. Molemmista rajapinnoista jäi puuttumaan ominaisuuksia, mutta tärkeimmät ominaisuudet saatiin kehitettyä. UART-rajapinnalle ehdotettiin jatkokehityksenä rajapinnan toteutusta ilman LibMF-kirjastoja, mikä mahdollistaisi UART-keskeytyksien käytön sekä vaihtoehtoisia toimintamalleja. GPIO-rajapinnalle ehdotettiin jatkokehityksenä puuttumaan jääneiden toimintojen, kuten analogisten signaalien käsittelyn, kehitystä. UART- ja GPIO-rajapintojen kehityksen ohessa tutustuttiin testivetoiseen kehitykseen. Järjestelmäpiirin hallitsemiseksi määriteltiin komentosyntaksi, jonka perusteella kehitettiin komentorajapinta. Komentorajapinta toteuttaa sille välitetyn viestin tulkinnan ja viestiä vastaavan toiminnon suorittamisen.

Testivetoisen kehityksen käyttäminen komentorajapinnan kehityksessä oli haastavaa, sillä TDD:n käytöstä ei ollut aiempaa kokemusta. Testivetoiseen kehitykseen tutustuessa selvisi, että TDD:stä hyötyminen vaatii kehitysmallin syvällistä ja kurinalaista opettelua, mikä saattaa kestää pidemmän aikaa. Siitä huolimatta projektityön aikana huomattiin, että TDD:n periaatteiden soveltamisella oli merkittäviä hyötyjä komentorajapinnan kehityksessä. Kehitettyjen komponenttien toiminnasta saatu välitön palaute johti siihen, että virheitä löydettiin nopeasti.

TDD:n käyttö johti myös ongelmiin. Testiympäristössä luodun ohjelmakokonaisuuden tuominen mikro-ohjaimelle osoittautui vaikeaksi, sillä testiympäristössä ei osattu huomioida tarpeeksi mm. mikro-ohjaimelle saatavilla olevia kirjastoja. Ongelmien esiintyminen johtui enimmäkseen mikro-ohjaimen arkkitehtuurin ja käytetyn kääntäjän tuntemattomuudesta ja TDD:n käytön kokemattomuudesta.

Projektin alussa oletettiin, että TDD:n käyttö johtaa alusta lähtien toimivampaan ohjelmakoodiin. Myös virheiden jäljitykseen ja korjaamiseen käytetyn ajan oletettiin pienenevän. Oletusta ei voida tämän projektin osalta vahvistaa ainakaan sulautetun ohjelmiston kehityksessä. TDD:n hyötyjen saavuttaminen vaatii pidempiaikaista harjoittelua ja kohdelaitteiston parempaa tuntemusta. Vaikka TDD:n käytöstä saadut tulokset eivät tämän projektin osalta täyttäneet oletuksia, jäi TDD:stä silti positiivinen kuva. TDD on hyödyllinen taito, jonka opettelulla voi kehittyä ohjelmistokehittäjänä ja korottaa arvoaan työmarkkinoilla.

LÄHTEET

Agile Manifesto 2017. Principles behind the Agile Manifesto. Viitattu 22.5.2017 http://agilemanifesto.org/principles.html.

Arduino 2017. Arduino MKRFOX 1200. Viitattu 25.4.2017 https://store.arduino.cc/homepage/arduino-mkrfox1200.

Barber D. 2012. Why Test-driven Development? Viitattu 25.4.2017 http://derekbarber.ca/blog/2012/03/27/why-test-driven-development/.

Briodagh, K. 2016. Sigfox Delivers Ultra-low Cost IoT Modules for Mass Market Deployment. IoT Evolution. Viitattu 25.4.2017 http://www.iotevolutionworld.com/m2m/articles/426957-sigfox-delivers-ultra-low-cost-iot-modules-mass.htm.

Collin, J. & Saarelainen, A. 2016. Teollinen internet. Helsinki: Talentum.

Connected Finland 2016. Suomen ensimmäinen esineiden internetin mobiiliverkko avattiin 29.9.2016. Viitattu 21.4.2017 http://www.connectedfinland.fi/news-view.php?article=37&title=suomen-ensimm%C3%A4inen-esineiden-internetin-mobiiliverkko-avattiin-29.9.2016.

Ericsson 2016. Ericsson mobility report. On the pulse of the networked society. Viitattu 21.4.2017 https://www.ericsson.com/res/docs/2016/ericsson-mobility-report-2016.pdf.

Grenning J. 2011. Test-Driven Development for Embedded C. Raleigh, North Carolina: The Pragmatic Bookshelf.

Hill, S. 2015. The Pros and Cons of Test-Driven Development. Viitattu 25.4.2017 https://leantesting.com/test-driven-development/.

IAR Systems 2011. IAR C/C++ Compiler Reference Guide for the 8051 Microcontroller Architecture. Versio 5. Viitattu 19.4.2017 http://supp.iar.com/filespublic/updinfo/005876/ew/doc/EW8051_CompilerReference.pdf.

Jones, N. 2009. Minimizing memory use in embedded systems Tip #3 – Don't use printf(). Embedded Gurus Blogs. Stack Overflow. Viitattu 19.4. 2017 http://embeddedgurus.com/stack-overflow/2009/09/minimizing-memory-use-in-embedded-systems-tip-3-dont-use-printf/.

Kernighan, B. & Ritchie, D. 1988. The C Programming Language. 2. painos. New Jersey: Prentice Hall PTR.

Martin, R. 2005. The Three Rules of TDD. Viitattu 26.4.2017 http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd.

Martin, R. 2014. The Cycles of TDD. The Clean Code Blog. Viitattu 26.4.2017 http://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html.

ON Semiconductor 2015. SoC Ultra-Low Power RF-Microcontroller for RF Carrier Frequencies in the Range 27 - 1050 MHz. Versio 3. Viitattu 6.4.2017 http://www.onsemi.com/pub/Collateral/AX8052F143-D.PDF.

ON Semiconductor 2016a. AX-SIGFOX-API SIGFOX Compliant Software Stack. Versio 2. Viitattu 19.4.2017 http://www.onsemi.com/pub/Collateral/AND9478-D.PDF.

ON Semiconductor 2016b. AX8052 Debugger Software Manual. Versio 3. Viitattu 19.4.2017 https://www.onsemi.com/pub/Collateral/AND9370-D.PDF.

ON Semiconductor 2016c. AX8052 LibMF Support Library Software Manual. Versio 3. Viitattu 19.4.2017 http://www.onsemi.com/pub_link/Collateral/AND9382-D.PDF.

ON Semiconductor 2016d. AX8052 Programming Manual. Versio 3. Viitattu 19.4.2017 http://www.onsemi.com/pub_link/Collateral/AND9349-D.PDF.

ON Semiconductor 2016e. Ultra-Low Power, AT Command / API Controlled, Sigfox Compliant Transceiver IC for Up-Link and Down-Link. Versio 4. Viitattu 19.4.2017 http://www.onsemi.com/pub/Collateral/AX-SFEU-D.PDF.

ON Semiconductor 2017. DVK-SFEU-1-GEVK: Sigfox Development Kit. Viitattu 24.4.2017 http://www.onsemi.com/PowerSolutions/evalBoard.do?id=DVK-SFEU-1-GEVK.

Palermo J. 2006. Guidelines for Test-Driven Development. Microsoft Developer Network. Viitattu 25.4.2017 https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx.

SIGFOX 2016a. Sigfox Core Service. Viitattu 20.4.2017 https://www.youtube.com/watch?v=6ZBGDtmDGRU.

SIGFOX 2016b. Lightweight Protocol For Small Messages. Viitattu 20.4.2017 https://www.youtube.com/watch?v=mS-6n5yAjR0.

SIGFOX 2016c. Introduction to the Sigfox LPWAN Low Power Wide Area Network. Viitattu 20.4.2017 https://www.youtube.com/watch?v=Xg_T8Q2W-OU.

SIGFOX 2016d. How A High Network Capacity Is Possible? Viitattu 20.4.2017 https://www.youtube.com/watch?v=oGeZpyCOnbc.

SIGFOX 2016e. Sigfox Ready certification for end products – Overview. Viitattu 24.4.2017 https://www.youtube.com/watch?v=6yp70wLHm98.

SIGFOX 2017. Sending a message. Viitattu 20.4.2017 http://makers.sigfox.com/about/.

SIGFOX 2017b. Our Story. Viitattu 20.4.2017 www.sigfox.com > About us > Our Story.

SIGFOX 2017c. Security of the Sigfox Network. Viitattu 21.4.2017 https://www.youtube.com/watch?v=zUUmOVlr1pQ.

SIGFOX 2017d. Coverage. Viitattu 21.4.2017 http://sigfox.com/en/coverage.