TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIIKA SUOMALAINEN

MIKROKONTROLLERIN SOVELTAMINEN MITTAUSDATAN LANGATTOMASSA TIEDONSIIRROSSA

Kandidaatintyö

TIIVISTELMÄ

MIIKA SUOMALAINEN: Mikrokontrollerin soveltaminen mittausdatan langattomassa tiedonsiirrossa
Tampereen teknillinen yliopisto
Kandidaatintyö, 20 sivua, 9 liitesivua
Joulukuu 2015
Automaatiotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Hydrauliikka ja automatiikka
Tarkastaja:

Avainsanat: kandidaatintyö, mikrokontrolleri, langaton, verkko, palvelin

Kandidaatintyössä käsitellään kuluttajamarkkinoilta saatavissa olevan mikrokontrollerin soveltuvuutta langattomaan tiedonsiirtoon. Käytännössä tässä kandidaatintyössä luodaan toimiva sovellus, joka siirtää anturin mittamaa dataa langattomasti palvelimelle ja esittää kyseisen datan käyttäjälle. Applikaatio koostuu mikrokontrollerista, Wi-Fi-ohjainmodulista sekä palvelimesta. Työ suoritettiin rakentamalla kytkennät fyysisten komponenttien välille ja luomalla ohjelmakoodi niiden ohjaamiseen. Ohjelmakoodista tehtiin mahdollisimman helposti ymmärrettävä mahdollista jatkokehitystä varten. Käyttäjän antamien muuttujien perusteella sovellus kytkeytyy langattomaan verkkoon ja lähettää mikrokontrollerilla sijaitsevan anturin mittaamaa dataa kyseisen verkon palvelimelle. Sovelluksen testaaminen toteutettiin kotiolosuhteissa tavallista, kuluttajakäyttöön suunniteltua, reititintä käyttäen.

Aineistona hyödynnettiin aihealueiden kirjallisuudesta ja tutkimuksesta löytyneitä artikkeleita, element14-yhteisön luomaa dokumentaatiota, Node.js-sivuston tarjoamaa dokumentaatiota ja mikrokontrollerin omaa dokumentaatiota. Käytettäviä tekniikoita valittaessa pääkriteereinä oli helppokäyttöisyys, reaaliaikaisuus sekä avoimuus. Työssä käytetyt ohjelmat ovat saatavilla verkosta maksutta.

Työssä käydään ensin läpi teoria sovelluksen takana. Teoria koostuu mikrokontrollerista, MEMS-anturista ja langattoman verkon tiedonsiirrosta. Teoriaa hyödynnetään työn toisessa osiossa, jossa käydään läpi sovelluksen toimintaa käytännössä. Käytännön osuudessa esitellään sovelluksen kytkennät sekä ohjelmointi.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

A/D	Analog/Digital
API	Application Programming Interface
ARM	Acorn RISC Machine
COM	Communication port
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IP	Internet Protocol
MEMS	Micro Electro Mechanical Systems
OSI	Open Systems Interconnection
PC	Personal Computer
SPI	Serial Peripheral Interface
SSID	Service Set Identifier
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protokol
USB	Universal Serial Bus
Wi-Fi	Kaupallinen nimike langattomille lähiverkoille.
WLAN	Wireless Area Network

1. JOHDANTO

Kandidaatintyön aiheena on mikrokontrollerin soveltaminen mittausdatan langattomaan tiedonsiirtoon. Työssä luodaan sulautettu järjestelmä, joka lähettää anturilta luettua dataa langattomasti tietokoneella sijaitsevalle palvelimelle. Valmistuneen ohjelman pohjalta oli tarkoitus luoda Mathworksin tarjoamaan simulink-ohjelmistoon yksinkertaiset ohjelmalohkot tiedonsiirtoa varten. Ajanpuutteen sekä käytetyn väliohjelmiston ja ohjelmistorajapinnan yhteensopimattomuuden vuoksi tähän tavoitteeseen ei kuitenkaan päästy.

Työ on jaettu kahteen pääosaan. Ensimmäisessä osassa esitellään työhön liittyvien aihealueiden teoriapohjaa. Mikrokontrolleri, serveri sekä verkkoprotokolla ovat oleellisia työn toisessa, käytännön osuudessa. Käytännön osuudessa käydään läpi fyysisten komponenttien kytkennät, mikrokontrolleria ohjaava koodi sekä palvelimella sijaitseva ohjelma. Osuuden lopussa esitellään myös ideoita, kuinka sovellusta voitaisiin kehittää toimimaan simulink-ympäristössä.

Applikaatio koostuu mikrokontrollerista, Wi-Fi-ohjainmoduulista sekä palvelimesta. Käytännössä tässä kandidaatintyössä luodaan toimiva sovellus, joka lukee anturin tarjoamaa mittausdataa ja lähettää sen langatonta verkkoa hyväksikäyttäen palvelimelle. Palvelimella vastaanotettu mittausdata tarjotaan käyttäjän nähtäväksi verkkoselainympäristöön. Työn painopisteenä on mikrokontrollerin toiminta. Palvelimen luominen oli käytännössä pakollista sovelluksen toiminnan testausta varten.

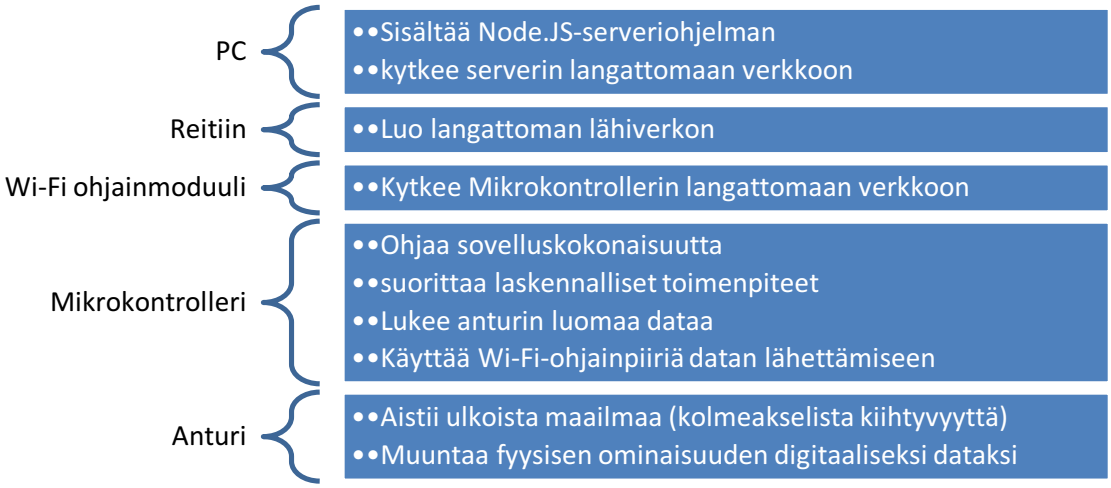
Mikrokontrollerin ohjelman kirjoittaminen suoritettiin Keil-µVision -kehitysympäristössä. Keil tarjoaa mittavan määrän työkaluja juuri mikrokontrollerien ohjelmoimista varten. Palvelimelle valittiinNode.js-ympäristö sen helppokäyttöisyyden vuoksi. Kumpikin käytetyistä ohjelmista on saatavissa verkosta ilmaiseksi.

2. SOVELLUKSEN TEORIA

Sovelluksen teoria-osiossa käydään läpi työlle oleellisimmat tekniikat ja protokollat. Näihin kuuluu mikrokontrollerin, anturin sekä langattoman lähiverkon toiminta. Työssä olennaisena osana käytettyä tiedonsiirtoprotokollaa tarkastellaan tarkemmin sen abstraktiuden vuoksi. Teorian ymmärrys auttaa ymmärtämään käytännön sovellusta.

2.1 Käytetyt tekniikat

Tässä kappaleessa on esitelty sovelluksen kannalta olennaisia tekniikoita ja protokollia. Käytettyjen komponenttien ja tekniikoiden tunteminen on olennaista sovelluksen toiminnan ymmärtämiseksi. Kuvassa 1 on esitelty sovelluksen peruskomponentit ja niiden tehtävät.



Kuva 1. *Työssä käytettyjen komponenttien pääasialliset tehtävät.*

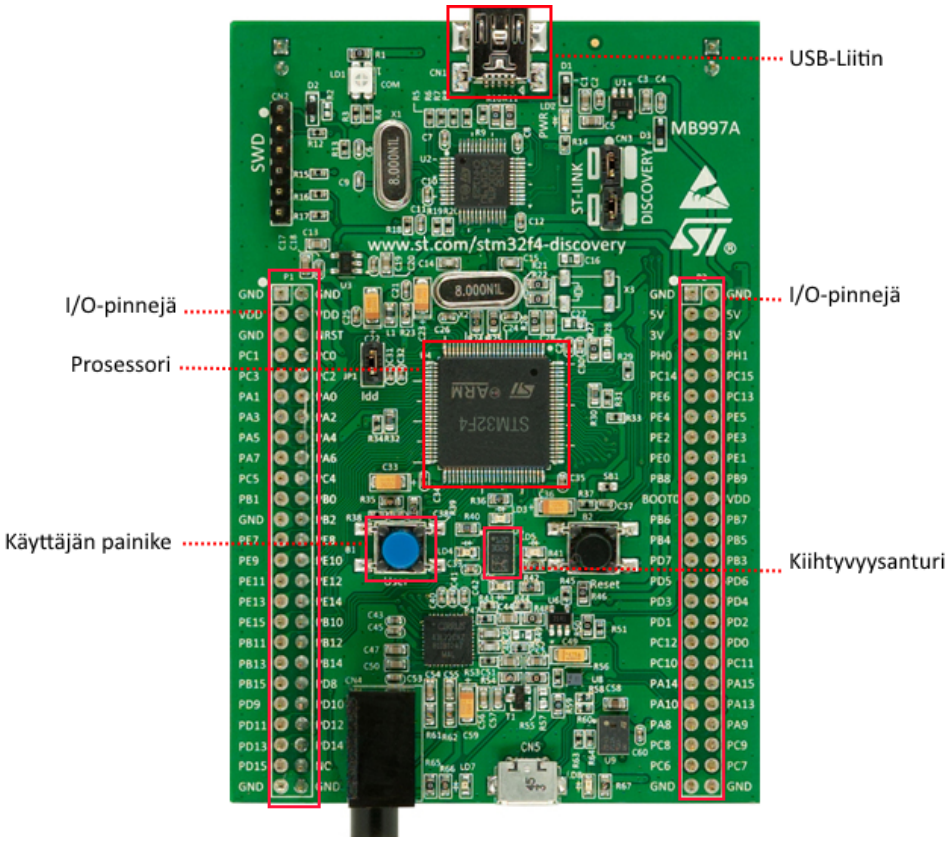
Mikrokontrolleri on sovelluksen oleellisin osa. Se sisältää suoritettavan ohjelman sekä ohjaa muita sovelluksen laitteita.

2.1.1 Mikrokontrollerit

1980-luvulta lähtien mikrokontrollerit ovat korvanneet monia aiemmin digitaalisilla logiikkapiireillä toteutettuja järjestelmiä. Mikro-ohjainpiirillä toteutettuja sovelluksia löytyy esimerkiksi autoista, leluista sekä kodinkoneista. Mikrokomponenttien hinnanlasku viime vuosikymmeninä on tehnyt niistä kustannustehokkaan tavan toteuttaa sulautettuja sovelluksia. Suosion kasvaessa ovat kontrollerivalmistajat kehittäneet useita lii-

tännäisohjainpiirejä. Näillä piireillä saadaan perusominaisuuksia laajennettua yksinkertaisesti. Mikroprosessoriin voidaan liittää esimerkiksi näyttöjä, kameroita sekä muita ulkoisia toimilaitteita. [1, s. 100]

Mikrokontrolleri on yksinkertaistettuna kokonaisuus, joka sisältää kaiken mikrotietokoneen vaatiman elektroniikan yhdellä piirillä. Yleensä piiri sisältää ainakin I/O(In/Out)-pinnejä, mikroprosessorin sekä erilaisia muistilohkoja. I/O-pinnien avulla kontrolleri on yhteydessä ohjattaviin laitteisiin. Näiden pinnien kautta kontrolleri myös lukee informaatiota muilta laitteilta, kuten antureilta. Mikroprosessorista löytyy ainakin aritmeettisen logiikan yksikkö sekä muistia. Siinä tapahtuu tiedon käsittely ja laskentatoimet. [2, s. 5–9] Prosessorin nopeutta kuvaa kellotaajuus, joka kertoo kuinka monta tilavaihdosta tapahtuu sekunnissa. Mikrokontrollereissa voi olla kolmea eri tyypistä muistia. Muistityypit ovat ohjelmamuisti, käyttömuisti sekä haihtumaton käyttömuisti. Ohjelma- sekä käyttömuisti ovat kontrollerin käytön kannalta välttämättömät. Ohjelmamuistissa sijaitsee mikrokontrolleriin ennalta asetettu ohjelma. Ohjelma on sarja ohjelmointikielen käskyjä, joiden perusteella kontrolleri ohjaa omaa sekä I/O-pinnien tilaa. Käyttömuistiin voidaan tallentaa ohjelman edetessä syntyvien muuttujien arvoja. Oleellisin ero ohjelma- sekä käyttömuistille on käyttömuistin tyhjentyminen käyttövirran katkaisun yhteydessä. Mikäli käyttömuistiin talletettujen muuttujien säilyttäminen on oleellista sovelluksen käytön kannalta, voidaan käyttää haihtumatonta käyttömuistia. [1, s. 102]



Kuva 2. *Sovelluksessa käytetty STM32F407 mikrokontrolleri[3].*

Mikrokontrollerit jaetaan kahteen pääryhmään niiden ohjelmoitavuuden mukaan. Sisäinen ohjelmamuisti voi olla joko valmistuksen yhteydessä kertaohjelmoitavaa tai uudelleenohjelmoitavaa. Näistä kahdesta kertaohjelmoitavat kontrollerit ovat selvästi harvinaisempia ja vaativat suuria tuotantomääriä, jotta niiden tuottaminen olisi kannattavaa. Suurin osa kuluttajamarkkinoilla olevista mikrokontrollereista on ohjelmoitavista tavallisella PC:llä (Personal Computer). Yleisimmille mikrokontrollereille on saatavilla useita eri kehitystyökaluja, jotka kääntävät käyttäjän syöttämän ohjelmointikieliset käskyt konekielelle ja lataavat ohjelman ohjelmamuistille. Ohjelmointikielinä voi toimia esimerkiksi C++, Python tai Java. [1, s. 102]

2.1.2 Älykkäät anturit

Anturilla tarkoitetaan laitetta, jonka tehtävä on muuntaa mitattava prosessisuure käyttökelpoiseksi sähköiseksi viestiksi. Anturin tuntoelin havaitsee mitattavan suureen ja anturiosa muuttaa suureen sähköiseksi. Sisäistä informaationkäsittelyä sisältäviä antureita kutsutaan älykkäiksi antureiksi. Älykkäät anturit sisältävät usein päätöksentekologiikkaa, laskentakapasiteettia sekä A/D(Analog/Digital)-muuntimia. Älykkäiden anturien eduiksi voidaan luetella muun muassa ohjelmalla muutettavissa oleva mitta-alue, pienemmät mittausvirheet, itsediagnostiikka sekä digitaalinen tiedonsiirto anturin ja ulkoisen toimilaitteen välillä. [4, s. 6–7] Valmistajan on mahdollista ohjelmoida anturin muistiin signaalia muokkaavia linearisointialgoritmeja. Algoritmien ansiosta voidaan poistaa esimerkiksi kohinaa ennen datan syöttämistä itse järjestelmään.

Älykkäiden antureiden valmistuksessa käytettävistä, noin 50-200mm leveät ja puoli millimetriä paksuista piikiteistä, saadaan valmistettua satoja keskenään samanlaisia piirejä. Pienen kokonsa vuoksi niitä kutsutaan usein mikroantureiksi, eli MEMS(Micro Electro Mechanical Systems)-antureiksi. [5, s. 19]

MEMS-anturissa anturin toiminnot on toteutettu monitoiminnallisilla polymeereillä kuten nanokokoisilla hiilikuituputkilla, piezoelektroniikalla ja muilla aktiivisilla keraameilla. Puolijohdetekniikan kehittyminen on mahdollistanut komponenttien pienentymisen mikrokokoon sekä tuonut mukanaan keinon sisällyttää anturiin elektroniikkapiirejä signaalinkäsittelyn tueksi. [6] Teknisten ominaisuuksien, pienen koon sekä luotettavuuden, ansiosta ovat mikrokoon anturit yleistyneet markkinoilla. Suurta valmistusvolyymia vaativat sovellukset, kuten autojen ilmatyynyjen anturisovellukset, ovat lähes aina toteutettu mikrokoon antureilla. [7, s. 101] Markkinoilla on tarjolla useita mikroantureita esimerkiksi kiihtyvyyden, asennon, paineen sekä voiman mittaukseen. Tässä työssä MEMS-anturia hyödynnetään kiihtyvyyden mittauksessa.

2.1.3 Langaton lähiverkko

Langattoman verkon avulla laitteet voivat viestiä keskenään ilman fyysistä tietoverkkoa. Tämä mahdollistaa laitteiden vapaan liikkumisen verkon vaikutusalueella. Ratkaisu vähentää myös oleellisesti tarvittavan kaapeloinnin määrää. Langattomat verkot voidaan jakaa ryhmiin sen perusteella kuinka laajan fyysisen alueen ne kykenevät peittämään. Sovelluksessa on käytetty langatonta lähiverkkoa jonka peittoalue kattaa yleensä yhden rakennuksen sisäiset tilat. [8, s. 3–5]

Langattoman verkon infrastruktuurin rakenneosa, joka yhdistää ilmateitse siirtyvän langattoman viestin lankaverkkoon, on nimeltään tukiasema. Käyttäjän tietokonelaiteessa oleva verkkokortti muodostaa yhteyden tukiasemaan, joka tarjoaa rajapinnan infrastuktuurin sisältämiin järjestelmiin ja verkon muihin tukiasemiin assosioituneisiin käyttäjiin. Sovelluksessa käytetty reititin on tukiaseman kehittyneempi versio. Se mahdollistaa usean tietokoneen toiminnan yhden ja saman laajakaista yhteyden kautta.[8 s38-39] Langaton lähiverkko on mahdollista luoda myös ilman reititintä käyttämällä esimerkiksi langatonta adhoc-tekniikkaa. Langattomien lähiverkkojen tunnistamiseen käytetään verkkotunnusta. Verkkotunnusta kutsutaan usein englannin kielestä johdetulla lyhenteellä SSID(Service Set Identifier). Sen avulla erotetaan samalla alueella olevat langattomat WLAN(Wireless Local Area Network)-verkot toisistaan. Verkkotunnukseen on mahdollista liittää myös salasana, jolla estetään ulkopuoliset yhteydet verkkoon. [9, s. 50–51]

Kuvassa 3 on esitetty verkon arkkitehtuurin OSI(Open Systems Interconnection Reference)-malli. Malli koostuu yhteensä seitsemästä kerroksesta ja se kuvaa langattoman verkon eri standardeja ja yhteentoimivuuksia. [8, s. 52]

Kuva 3. *Verkon OSI-malli.*

Työssä esiteltävän sovelluksen kannalta oleellisimmat kerrokset ovat fyysinen, siirtoyhteys-, verkko- sekä kuljetuskerros. Fyysinen kerros sisältää radioaallot ja infrapunavalon, joiden avulla käytännössä fyysinen tiedonsiirto tapahtuu. Siirtoyhteyskerros varmistaa pääsyn siirtotiehen sekä kahden yksikön välisen synkronoinnin ja virheiden valvonnan. Verkkokerros suorittaa pakettien reitityksen verkossa lähteestä kohteeseen.

Tässä kerroksessa toimii muun muassa myöhemmin esitelty IP(Internet Protocol)-protokolla. Kuljetuskerros tarjoaa mekanismin virtuaaliyhteyksien muodostamiseen ja ylläpitoon. Tässä kerroksessa toimii muun muassa sovelluksessa käytetty TCP(Transmission Control Protocol)-protokolla. [8, s. 52–53]

Verkkoarkkitehtuurin kerrokset määrittelevät yhdessä langattoman verkon ominaisuudet. Langaton verkkokortti toteuttavat itsenäisesti vain mallin kahden alimman kerroksen toiminnallisuudet. Verkon muut kerrokset toteutetaan väliohjelmistoilla. [8, s. 53]

Teollisen internetin sekä langattomien antureiden yleistyessä tulee energiankulutukseen kiinnittää erityistä huomiota. Tästä syystä langattomien verkkojen kehitykseen on asetettu viimevuosina paljon resursseja. Työssä käytetty IEEE 802.11n(Wi-Fi)-tekniikka mahdollistaa lähiverkolle suuren tiedonsiirtonopeuden sekä kantaman, mutta energiatehokkuutta priorisoitaessa, olisi suotavaa selvittää muita mahdollisia verkkotekniikoita. Varteenotettava vaihtoehto voisi olla esimerkiksi kotiautomaatiossa käytetty Z-wave-teknologia.

2.1.4 Tiedonsiirtoprotokolla TCP/IP

TCP/IP-protokollaperhe mahdollistaa tietokoneiden, matkapuhelimien sekä muiden sitä tukevien sulautettujen järjestelmien keskinäisen kommunikoinnin verkon välityksellä. Protokolla on täysin avoin eikä se ole riippuvainen käytetystä ohjelmistosta eikä laitevalmistajasta. Sen vuoksi se toimii maailmanlaajuisesti internetin perustana. [10, s. 2]

Kun dataa lähetetään palvelimelta toiselle IP-protokollaa käyttäen, jakaa protokolla datan pienemmiksi, helpommin käsiteltäviksi, ”paketeiksi”. Jokaiselle paketille asetetaan ylätunnus, johon sisällytetään osoite ja reititystietoa. Tämä mahdollistaa vastaanottavan palvelimen koota saapuvat paketit yhteen ja muodostaa alkuperäinen data. [11, s. 20] TCP-protokollan tehtävä on taata pakettien siirtyminen sekä pakettien eheys. Mikäli pakettien perillemenon takaaminen ei ole välttämätöntä, voidaan käyttää kevyempää UDP(User Datagram Protokol)-protokollaa.

Kun kahden laitteen IP-osoitteet tunnetaan, voidaan niiden välille muodostaa tiedonsiirtoyhteys TCP-protokollaa käyttäen. TCP pystyy takaamaan datan perillemenon ja korjaamaan siirtovirheet, jotta sovelluksen ei ole tarpeen huolehtia kyseisistä asioista. TCP-yhteys muodostetaan päätepisteiden välillä ennen varsinaisen tiedonsiirron aloittamista. Näin ollen molemmilla päätepisteillä on myös mahdollisuus vaikuttaa yhteyden sulkemiseen. TCP-protokollan ensisijainen tehtävä on käsitellä tavuvirtaa kahden päätepisteen välisessä tiedonsiirrossa. TCP-protokolla myös valvoo IP-protokollan toimintaa, sekä käyttää sitä yhteyksien muodostamiseen. Päätepisteissä on lähettävä ja vastaanot-

tava sovellus. Lähetettävä ja vastaanottava sovellus tunnistetaan porttinumeroiden avulla. [12, s. 166]

Soketilla tarkoitetaan IP-osoitteen ja TCP:n porttinumeron muodostamaa paria. Näillä tiedoilla voidaan yksikäsitteisesti tunnistaa verkkoa käyttävä sovellus. Portit ovat tietokoneen sisäisiä 16-bittisiä kokonaislukuja. Mahdollisia porttinumeroita ovat siis luvut väliltä 0-65535. Osa näistä porttinumeroista on ennalta varattu tiettyjen sovellusten käyttöön. [12, s. 194]

2.2 Käytetyt komponentit ja ohjelmointiympäristöt

Tässä kappaleessa on esitelty käytettyjen komponenttien ominaisuuksia ja tehtäviä sulautetussa sovelluksessa. Mikrokontrollereita on markkinoilla valtava määrä ja niiden ominaisuudet vaihtelevat suuresti. On siis tärkeää spesifioida fyysisten komponenttien olennaisimmat ominaisuudet. Myös ohjelmointiympäristön valinta vaikuttaa lopullisen käytännön sovelluksen toimintaan.

2.2.1 Komponentit

Työssä käytetty mikrokontrolleri on STMicroelectronicsin valmistama STM32F407-Discovery. Mikrokontrolleri on ns. evaluointilauta, jolla on helppo tutustua tuoteperheen ominaisuuksiin ja valita lopulliseen sovellukseen oikeat komponentit. Kontrolleri sisältää 168Mhz kellotaajuudella toimivan prosessorin sekä 1Mt flash muistia. Mikrokontrolleriin on integroitu myös työssä hyödynnetty MEMS-kiihtyvyysanturi, 140 I/O-pinniä eri toiminnallisuuksilla, käyttäjän ohjelmoitavissa oleva painike sekä ledejä. Jokainen kontrollerille integroitu komponentti on yhdistetty sen käyttämiin I/O-liittimiin. Näitä komponentteja käsitellään ohjelmatasolla, kuin ne olisivat ulkoisia komponentteja. Mikrokontrolleri sisältää useita erilaisia tiedonsiirtorajapintoja.[13] Anturin ja mikrokontrollerin välisenä rajapintana käytetään SPI(Serial Peripherial Interface)-rajapintaa. Ohjelmointi-PC:n ja mikrokontrollerin sekä mikrokontrollerin ja Wi-Fi-ohjainpiirin välinen tiedonsiirto on toteutettu UART(Universal Asynchronous receiver Transmitter)-rajapinnalla.

STM32F407 Wi-Fi-ohjainpiiri on itsenäiseen toimintaan kykenevä sekä sertifioitu verkkokontrollerimoduuli, joka mahdollistaa STM32F407-mikrokontrollerin kytkemisen langattomaan verkkoon. Moduuli on mahdollista yhdistää mikrokontrolleriin sarjamuotoisella rajapinnalla, kuten UART:illa tai SPI:llä. Ohjainpiiri on Embest Technology Co:n suunnittelema ja sen toiminta pohjautuu Muratan SN8200 Wi-Fi-verkkomoduuliin. STM32F407 Wi-Fi-ohjainpiiri tukee TCP/IP protokollaperhettä sekä useita muita verkkostandardeja. [14]

Kolmas fyysinen komponentti on Silicon Labs:in valmistama CP2102 USB(Universal Serial Bus)-UART muunnin. Komponentti sisältää sisäistä ohjelmoitavaa muistia, jännite-regulaattorin, kello-oskilaattorin sekä I/O-liittimiä. Komponentin tehtävä on luoda virtuaalinen sarjaliikenneportti USB-porttia hyväksikäyttäen. CP2102 tarjoaa liitännät datan lähetykseen(TxD), vastaanottamiseen(RxD), 3.3V tai 5V käyttöjännitteeseen sekä maadoitukseen [15]. Kun komponentti kytketään USB-porttiin, tunnistaa tietokone sen COM(Communication Port)-porttina. Porttia voidaan lukea pääte-emulaattorilla. [15] Työssä käytetään PuTTY-nimistä emulaattoria.

2.2.2 Ohjelmointiympäristöt

Keil tarjoaa tehokkaan ohjelmointiympäristön joka on räätälöity mikrokontrollien ohjelmoimiseen. Se tukee yli tuhatta ARM(Advanced RISC Machines)-arkitehtuurin mikroprosessoreihin pohjautuvaa mikrokontrolleria. Ohjelmointiympäristöstä on mahdollista asettaa käytettävä kontrolleri. Tämän perusteella ladataan perusohjelmakirjastot ja asetetaan kontrollerin perusasetukset. Käyttäjä saa näkyviinsä vain kyseiselle laitteelle mahdolliset ominaisuudet, mikä helpottaa asetuksien muokkaamista. Ohjelma sisältää tekstieditorin, ohjelmointikielen kääntäjän, Debuggerin sekä monia muita ohjelmointia helpottavia ominaisuuksia. [16]

Node.js on palvelinpuolen ohjelmointiympäristö. Sen ohjelmointikielenä toimii JavaScript-ohjelmointikieli. Poiketen monista muista palvelinratkaisuista, Node-palvelimen ei tarvitse avata erillisiä säikeitä jokaiselle yhteydelle. Tämän vuoksi Nodella toteutettu palvelin kuluttaa paljon vähemmän resursseja, kuten esimerkiksi suosittu Apache-palvelinohjelmisto. Perinteisistä palvelinsovelluksista node.js poikkea myös siten, että se on vahvasti tapahtumiin pohjautuva. Tapahtumakeskeisessä ohjelmassa tietyt tapahtumat, kuten hiiren klikkaukset, yhteydenottopyynnöt tai datan saapuminen, voivat käynnistää ennalta määrätyn osan ohjelmasta. [17] Node.js ohjelmointiympäristöllä on mahdollista toteuttaa monipuolisia reaaliaikaiseen laskentaan pohjautuvia sovelluksia. [18]

3. KÄYTÄNNÖN SOVELLUS

Kolmannessa kappaleessa tarkastelemme teoriapohjaa hyödyntäen lopullista sovellusta ja sen toimintaa. Ensimmäisenä suunnittelutavoitteena oli luoda helppokäyttöinen ja muokattavissa oleva sovellus hyödyntäen STM32F407-mikrokontrolleria sekä siihen suunniteltua Wi-Fi-ohjainpiiriä. Sovelluksen pohjalta oli tarkoitus luoda Simulink-lohko yksinkertaistamaan mikrokontrollerin ohjelmointia. Ensin käydään läpi mikropiirien välille luodut kytkennät. Seuraavaksi tarkastellaan mikrokontrollerilla ja serverillä olevaa ohjelmakoodia. Ohjelmakoodista on poimittu oleellisimmat osat ja käyty läpi niiden tehtävä kokonaisuutta suoritettaessa. Lopuksi käydään läpi ideoita kuinka tulisi toimia sovelluksen mahdollisessa jatkokehityksessä.

3.1 Komponenttien kytkennät

Sovelluksen kehittäminen aloitettiin kytkentöjen suunnittelulla. Mikrokontrollerin sekä Wi-Fi-ohjainpiirin dokumentaatioista etsittiin kytkentäpinnit UART-sarjaporttiyhteyden muodostamiseen. Mikrokontrollerista etsittiin myös UART-kytkentäpinnit, joiden avulla luotiin sarjaliikenneyhteys tietokoneeseen[13, s. 46–58][14, s. 9]. Toteutettu kytkentä on esitetty taulukossa 1.

Taulukko 1. *Sovellukseen toteutetut kytkennät*

STM32F407 Mikrokontrolleri	**Wi-Fi-ohjainpiiri**
PB6 (USART1_TX)	J6 – pinni 3 (USART_RX)
PB7 (USART1_RX)	J6 – pinni 5 (USART_TX)
GND	J6 – pinni 11 (GND)
GND	J6 – pinni 12 (GND)
STM32F407 Mikrokontrolleri	**CP2102 USB-UART**
PC10 (USART3_TX)	USART_RX
PC11 (USART3_RX)	USART_TX
GND	GND

Taulukossa esitellään mikrokontrollerin ja Wi-Fi-ohjainpiirin, sekä mikrokontrollerin ja UART-USB-muuntimen välille muodostetut kytkennät. Ohjainpiiriin tulee neljä johdinta ja UART-USB-muuntimeen kolme.

3.2 Mikrokontrollerin ohjelmoiminen

Mikrokontrollerin ohjelmoimisen osio on jaettu neljään osioon. Ensimmäisessä osiossa käydään läpi sovelluksen kehittämisen perustiedon, kuten käytetyt ohjelmakirjastot ja ohjelmointiympäristö. Seuraavissa osioissa käsitellään Wi-Fi- ja TCP-yhteyden muodostaminen. Lopuksi tarkastellaan anturin toimintaa sekä tiedon lähetyksessä käytettyjä funktioita.

3.2.1 Asetukset ja sovellukset

Mikrokontrollerin ohjelmoiminen aloitettiin lataamalla Wi-Fi-ohjausmodulin valmistajan laatima ohjelmointikirjastojoukko [19]. Käyttäjälle tarjolla oleva tiedosto sisältää valmiin projektin Keil-ohjelmointiympäristölle. Projekti esittelee ohjainpiirin perustoiminnallisuuksia. Työssä muokattiin tiettyjä funktioita haluttujen toimintojen luomiseksi. Alkuperäinen projekti on riippuvainen komentoriviemulaattorin kautta annetuista syötteistä läpi ohjelman etenemisen. Funktioiden muokkaamisella pyrittiin irroittautumaan aktiivisista syötteistä ja mahdollistamaan itsenäisen toiminnan ennalta ohjelmaan annettuja muuttujia hyödyntäen. Muuttujia ovat SSID-tunnus, salasana, suojaustaso, vastaanottavan laitteen IP-osoite sekä käytettävä porttinumero. Ohjelmaan lisätyt muuttujat on esitelty ohjelmassa 1. Valmistajan luomat funktiot jätettiin projektiin mahdollisen jatkokehityksen tarpeisiin.

```
//Wi-Fi asetukset
char WiFi_nimi[20]= "AirLink3G";
char turva[2]="4";
char salasana[20]= "salasana1234";

char IPosoite[20] = "192.168.0.104";
int kohdeportti1 = 9001;
int kohdeportti2 = 9002;
int kohdeportti3 = 9003;

int8_t kiihtyvyys1;
int8_t kiihtyvyys2;
int8_t kiihtyvyys3;

int8_t mysock1 = -1;
int8_t mysock2 = -1;
int8_t mysock3 = -1;
```

Ohjelma 1. *Työssä käytetyt muuttujat*

Ladattu projekti koostui kolmesta pääkirjastosta. Sn8200_hal-kirjasto on suorassa yhteydessä Wi-Fi-ohjauspiiriin. HAL-kirjasto toteuttaa sarjaportissa tapahtuvan liikenteen ja vastaanottaa tietoa korkeamman tason kirjastojen käsiteltäväksi. Kaksi korkeamman tason ohjelmointikirjastoa ovat Sn8200_core sekä Sn8200_API. Työssä keskitytään API(Application Programming Interface)-kirjaston muokkaamiseen, sillä se tarjoaa korkeimman tason ohjelmointirajapinnan käyttäjälle.

Mikrokontrollerin sekä ohjelmoinnissa käytetyn tietokoneen välille luotiin sarjaliikenneyhteys PuTTY-pääte-emulaattorilla. Yhteys helpottaa langattoman sovelluksen kehittämistä sekä toiminnan havainnollistamista.

3.2.2 Wi-Fi-yhteys

Wi-Fi-yhteyden luominen on toteutettu pääohjelmassa neljällä funktiolla. Ohjelma 2 esittelee käytetyt funktiot niiden suorittamisjärjestyksessä.

```
    //Muodostetaan Wi-fi-yhteys reitittimen kautta
2   // turva: 0 open, 2 WPA TKIP, 4 WPA2 AES, 6 WPA2 MIXED
    WifiDisconn(seqNo++);
4   WifiJoinOMA(seqNo++,WiFi_nimi,turva,salasana);
    SnicInit(seqNo++);
6   SnicIPConfig(seqNo++);
```

Ohjelma 2. *Wi-Fi-yhteyden luomisessa käytetyt funktiot*

Ensimmäinen funktio katkaisee mahdollisesti käytössä olevan yhteyden. Toinen WifiJoinOMA-funktio on ainoa johon on tehty muutoksia. Funktioon on lisätty kolme syötettä. Wifi_nimi ja salasana ovat verkolle ominaiset SSID-tunnukset sekä salasana. Turva-muuttuja valitsee käytettävän suojaustason. Funktio yhdistää laitteen haluttuun verkkoon. Kaksi viimeistä funktiota alustavat verkkokontrollerin sisäisiä tietoja.

Ohjelma 3 näyttää osan WifiJoinOMA-funktiosta. Ohjelmakoodissa on nähtävillä funktioon tehdyt muutokset.

```
printf( "Enter SSID:\n\r" );
strcpy(SSID,WiFi_nimi);    //Kopioidaan SSID
while(!strlen(SSID)) {
  printf( "SSID can't be empty. Enter SSID:\n\r" );
  scanf( "%s" , SSID);
  printf( "\n\r" );
}
memcpy(p, SSID, strlen(SSID));

p += strlen(SSID);
*p++ = 0x00;

printf( "Enter Security Mode (e.g., 0 for open, 2 for WPA TKIP, 4 for
WPA2 AES, 6 for WPA2 MIXED):\n\r" );
strcpy(tempstr,turva);       //Kopioidaan salaustason numero
printf( "\n\r" );
secMode = atoi(tempstr);

if (secMode) {
  printf( "Enter Security KeyOMA:\n\r" );
  strcpy(secKey,salasana); //Kopioidaan ennalta annettu salasana
  printf( "\n\r" );
  Keylen = (unsigned char)strlen(secKey);

  if (Keylen <= 0) {
    printf( "Invalid Key\n\r" );
    return;
    }
  }
```

Ohjelma 3. *WifiJoinOMA-funktio*

Ohjelmaan on lisätty strcpy-funktiot siirtämään käyttäjän antama Wifi-verkon tunnus, salasana sekä turvaustason numero järjestelmän käytettäväksi.

3.2.3 TCP-yhteys

Sovelluksessa käytettävä anturi antaa kiihtyvyystietoa x-, y- ja z-akselilla. Jokaisen akselin mittausdatalle luodaan oma sokettipari, joiden avulla dataa siirretään palvelimelle. Vastaanottava palvelin on ohjelmoitu kuuntelemaan aktiivisesti soketteja 9001,9002 sekä 9003.

Ohjelma 4 näyttää kuinka soketin ja TCP-yhteyden luominen on toteutettu. Kokonaisuuden ymmärrettävyyden vuoksi funktio on sijoitettu suoraan pääohjelmaan.

```
2  //luodaan ensimmainen socketti palvelimelle
       mysock = -1;
4      tcpCreateSocket(0, 0xFF, 0xFF, seqNo++, SNIC_TCP_CREATE_SOCKET_REQ);
       if (mysock != -1) {
6        printf("mysokin arvo: %d",mysock);
         mysock1 = mysock;
8        if (getTCPinfoOMA(kohdeportti1,IPosoite) == CMD_ERROR) {
           printf("Invalid Server\n\r");
10       }
         tcpConnectToServer  (mysock1,  destIP,  (unsigned  short)destPort,
12 0x0400, 0x5, seqNo++);
      }
14
```

Ohjelma 4. *Soketin luomiseen käytetyt komennot*

TcpCreateSocket-funktio luo funktion ja muuttaa mysock-muuttujan käytettävän portin numeroksi. Numero tallennetaan mysock1-muuttujaan myöhempää käyttöä varten. Soketin numeroa tarvitaan yhteyksiä suljettaessa.

GetTCPinfoOMA-funktio kerää tarvittavat tiedon TCP-yhtyden muodostamiseen. Funktiolle annetaan syötteinä kohdelaitteen IP-osoite sekä soketin porttinumero. Funktioon on tehty pieniä muutoksia. Muutokset on esitelty ohjelmassa 5. Funktion tarjoamia tietoja hyväksi käyttäen luo tcpconnectToServer-funktio lopulta TCP-yhteyden.

```
   printf("Enter server IP to connect: \n\r");
2  strcpy(tempIPstr,IPosoite);  //ennalta maaratty ip-osoite
   printf("\n\r");
4  if (strlen(tempIPstr))
     strcpy(IPstr, tempIPstr);
6  destIP = inet_addr(IPstr);
   if (destIP == INADDR_NONE || destIP == INADDR_ANY) {
8    return CMD_ERROR;
       }
10
   printf("Enter server port number: \n\r");
12 snprintf(teststr, sizeof(teststr), "%d", portti);
   //muutetaan funkion annettu arvo arrayksi
14 printf("\n\r");
```

Ohjelma 5. *Soketin luomiseen käytetyt komennot*

Strcpy- sekä snprintf-funktioilla siirretään syötteenä annetut muuttujat järjestelmän käyttämiin muuttujiin. Funktio suorittaa lopuksi virhetarkastelua.

3.2.4 Tiedon lähetys

Tiedon lähetys koostuu mittausdatan poimimisesta anturilta sekä kyseisen datan lähettämisestä serverille. Jotta anturitietoa voidaan kerätä, on anturin käyttämä SPI-väylä alustettava. Samalla alustetaan myös käyttäjän painike toimimaan halutulla tavalla. Alustusfunktiot, sekä SPI-väylän tiedonsiirrosta huolehtivat funktiot ovat kokonaisuudessaan nähtävillä liitteenä olevassa kooditiedostossa. Ohjelma 6 näyttää pääohjelmassa sijaitsevat alustuskomennot.

```
//alustetaan anturin SPI väylä
mySPI_Init();
MY_BUTTON_init(); //alustetaan kayttajan painike

mySPI_SendData(0x20, 0x67); //asetetaan ctrlreg_1 kaikki akselit kaytet-
tavaksi
mySPI_SendData(0x24, 0x00); //asetetaan tarkkuus 2G/4G/8G

```

Ohjelma 6. *Anturin sekä painikkeen alustus*

Anturin dokumentoinnista etsittiin rekisterit kiihtyvyysakselien sekä halutun skaalauksen aktivoimiseksi[20, s. 29–31]. Rekisterien muutos tapahtuu anturille lähetettävällä heksadesimaaliluvulla. Luku muutetaan desimaaliluvuksi, jonka arvo määrää rekisterin toiminnan.

Alustuksen jälkeen mikrokontrolleri on valmiina lähettämään dataa verkkoon. Lähetys aloitetaan mikrokontrollerista löytyvää painiketta painamalla. Lähetystoiminto on toteutettu while-silmukalla, joka jokaisen kierron alussa päivittää anturilta saatavat kolme arvoa. Anturi tarjoaa kutsuttaessa kiihtyvyystiedon int-tyyppisenä kokonaislukuna. Jotta tämä luku voidaan lähettää, on se muutettava char-vektoriksi. Tuotetun vektorin käsittelyä varten on luotu teststrOMA- sekä lenOMA-muuttujat. Näistä ensimmäiseen tallennetaan vektori. Jälkimmäisen avulla analysoidaan tuotetun vektorin pituutta. Ohjelmassa 7 on esitetty, kuinka anturilta kutsutaan yhden anturiakselin dataa ja miten selähetetään sokettia pitkin palvelimelle.

```
kiihtyvyys1 = mySPI_GetData(0x29);    //poimitaan kiihtyvyysdata

snprintf(teststrOMA, sizeof(teststrOMA), "%d", kiihtyvyys1);
printf("anturi antoi1: %d ",kiihtyvyys1);
lenOMA = (int)strlen(teststrOMA);
sendFromSock(mysock1, (int8u*)teststrOMA, lenOMA, 2, seqNo++);


```

Ohjelma 7. *Kiihtyvyysdatan kutsu anturilta sekä tiedon lähetys*

Tiedon poiminta ja lähetys toteutetaan jokaiselle kiihtyvyyssuunnalle erikseen. Tiedon lähetys jatkuu niin kauan kuin käyttäjä painaa mikrokontrollerista löytyvää painiketta uudestaan.

Kun tiedon lähetys lopetetaan, sulkeutuvat myös aikaisemmin luodut sokettiyhteydet. Tiedon lähetyksen lopettamisen yhteydessä tapahtuvat toimenpiteet on esitetty ohjelmassa 8.

```
//painikkeella lopetetaan lahetys ja suljetaan soketit
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)){
  lahetys = false;
  closeSocket(mysock1, seqNo++);
  closeSocket(mysock2, seqNo++);
  closeSocket(mysock3, seqNo++);
  mdelay(500);
}
```

Ohjelma 8. *Tiedon lähetyksen lopettaminen*

Mikäli tietoa halutaan lähettää uudestaan, on mikrokontrolleri käynnistettävä uudestaan reset-painikketta painamalla. Tämän jälkeen laite kytkeytyy uudestaan verkkoon ja luo soketit tiedonsiirtoa varten.

3.3 Serverin ohjelmoiminen

Työn kehittämisen mahdollistamiseksi oli luotava serveri, jolle mikroprosessorilla lähetettävä tieto vastaanotetaan. Serverin ohjelmoiminen aloitettiin etsimällä käyttötarkoitukseen paras sovellus. Node.js osoittautui parhaaksi vaihtoehdoksi sen yksinkertaisuuden ja reaaliaikaominaisuuksien vuoksi. Node.js on saatavilla valmistajan sivuilta [21]. Ohjelman kirjoittamiseen riittää tavallinen tekstieditori, jolla tuotettu koodi käännetään toimivaksi ohjelmaksi komentorivillä. Node.js toimii siis JavaScript-tulkkina.

Serverin perustoiminnallisuuksia ovat tiedon vastaanottaminen, sekä vastaanotetun tiedon vieminen html(Hyper Text Markup Language)-sivulle. Tältä html-sivulta on mahdollista seurata saapuvia tietoja. Sekä serverille, että html-sivulle on oma ohjelmakoodinsa.

Internet-selaimella tapahtuvaan yhteydenottoon reagoidaan lähettämällä html-sivu nimeltä index.html. Html-sivu lähetetään kun yhteydenotto tapahtuu porttiin numero 3000. Html-sivun lähettäminen http(Hypertext Transfer Protocol)-protokollaa käyttäen esitellään ohjelmassa 9.

```
//yhdistetaan kansiossa oleva html tiedosto "C:\palvelin\...
index = fs.readFileSync(__dirname + '/index.html');
console.log(__dirname);

// Lähetä index.html kaikkiin pyyntöihin.
var app = http.createServer(function(req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end(index); //lahetetaan indexi sivu yhteytta otettaessa
});
app.listen(3000); //html kuuntelee porttia 3000


```

Ohjelma 9. *Serverin toiminta http-yhteydenotossa*

Ohjelmakoodissa määritellään index-muuttuja sisältämän index.hml-nimisen sivun. Seuraavaksi määritellään app-funktio, joka reagoi yhteydenottoon lähettämällä index.html-sivun selaimelle.

Serveri on asetettu kuuntelemaan kolmea eri sokettia, joita hyväksikäyttäen on mahdollista luoda TCP-yhteys. Serverin toiminta soketin luomisen yhteydessä on esitetty ohjelmassa 10.

```
//Luodaan TCP serveri, joka odottaa yhteydenottoa
var socket1 = net.createServer(function(sock1){
  console.log('tcp1 yhteys muodostettu');
  sock1.setNoDelay(true);
  sock1.on('data', function(data) {  //kun dataa saapuu
    console.log('DATA1 tuli ' + socket1.remoteAddress + ': ' + data);
    io.sockets.emit('livedata1', 'data1:'+data);
  });

}).listen(9001);
```

Ohjelma 10. *Serverin toiminta TCP-yhteydenotossa*

Ohjelma ilmoittaa komentoriville kun yhteys on muodostettu. Tämän jälkeen serveri jää odottamaan saapuvaa dataa. Saapunut data tulostuu komentoriville sekä siirtyy index.html-sivulle.

Index.html-sivun ohjelma rakentuu kolmesta viestikentästä, jotka muuttavat arvoaan kun serveriltä saapuu dataa. Scriptissä on hyödynnetty socket.io-kirjastoa. Kyseessä on sama kirjasto jota käytetään serveriohjelmassa tiedon lähetykseen. Saapuvalla datalla on serveriltä lähetetty otsikko, jonka perusteella saapuva data tunnistetaan. Ohjelmasta 11 nähdään toimenpiteet tietoa vastaanotettaessa.

```
//tietoa saapuu livedata1 otsikolla
socket.on('livedata1', function(data) {
  console.log(data);
  $('#messages1').html(data);
});

```

Ohjelma 11. *Toimenpiteet datan saapuessa html-sivulle*

Kun dataa saapuu, ohjataan se sivun body-osiossa sijaitseviin muuttujiin. Nämä muuttujat ovat luettavissa internet-selaimella.

3.4 Jatkokehitys

Työn tavoitteena oli kehittää ohjelma, josta pienin muutoksin voitaisiin luoda Mathworksin kehittämään simulink-simulointiohjelmistoon omat lohkot langattoman yhteyden luomiseen [22]. Lohkojen kehityksessä käytettiin Aimagin-yhtiön tarjoamaa Waijung-lohkojoukkoa [23]. Waijung tarjoaa mahdollisuuden rakentaa simulink-lohkoja hyödyntäen omaa C-koodia.

Lohkoja luodessa huomattiin, että Waijung-lohkojoukon käyttämä kääntäjä ei kyennyt käsittelemään switch-case-rakennetta. Wi-Fi-moduulin kehittäjän tarjoama ohjelmointikirjasto Sn8200_core sisältää paljon tämän tyyppisiä rakenteita. Ongelma voidaan ratkaista ohjelmakirjaston rakenteiden uudelleenohjelmoinnilla, tai Waijung-lohkojoukon käyttämän kääntäjän vaihtamisella.

4. YHTEENVETO

Teollisen internetin, pilvipalveluiden ja langattomuuden käyttö kasvaa teollisuudessa jatkuvasti. Näiden tekniikoiden perussovelluksien rakentamiseen riittää ohjelmointitekniikoiden, sekä elektroniikkakomponenttien toiminnan perusymmärrys. Kuluttajille tarjolla olevat komponentit ovat halpoja ja tehokkaita. Ennen komponenttien valintaa on hyvä tutustua tarjolla olevaan dokumentaatioon, valmistajan tarjoamiin ohjelmointikirjastoihin sekä alan harrastajien aktiivisuuteen verkossa. Ohjelmaa rakentaessa, aiheutti valmistajan tarjoama ohjelmointikirjasto paljon ongelmia. Dokumentointi oli epäselvä ja valmiista projektista pienempien ominaisuuksien etsiminen kulutti paljon aikaa. Vaikka esimerkki oli käytännönläheinen, oli sitä työlästä lukea.

Keil-ohjelmointiympäristö on hyvä valinta kyseiseen sovellukseen. Keil tarjosi hyödyllisiä työkaluja sovelluksen kehittämiseen. Ongelmatilanteessa forumin kautta oli mahdollista saada vastauksia. Kun suuri osa asetuksista ja perusohjelmointikirjastojen asennuksesta on automatisoitu, voi käyttäjä keskittyä itse ohjelmoimiseen. Ohjelmointiympäristö teki myös virheiden etsimisen helpoksi.

Vaikka Node.js on suhteellisen tuore kirjasto, tarjosi se helposti omaksuttavan keinon kehittää yksinkertainen serverisovellus. Dokumentaatio oli selkeä ja verkossa toimiva yhteisö todella aktiivinen. Luotu serveri ei kuitenkaan aivan täyttänyt vaatimustasoa. Kun mikrokontrollerin lähetystaajuutta kasvatettiin tarpeeksi suureksi, ei serveri ehtinyt reagoida, vaan tulosti saapuneita merkkijonoja yhdistettynä. Tästä syystä ohjelman rivillä 334 jouduttiin lisäämään viive. Viiveen ansiosta järjestelmän testaus voitiin suorittaa olemassa olevalla palvelimella. On kuitenkin pidettävä mielessä, että mikrokontrolleri kykenee selvästi korkeampaan lähetysnopeuteen, kuin mitä tässä sovelluksessa käytetään.

LÄHTEET

[1] J. Koskinen. Mikrotietotekniikka - Sulautetut järjestelmät, Otava, 2004.

[2] J. Davies, MSP430 – Microcontroller basics, Elsevier, 2008.

[3] STMicroelectronics, STM32F407-Discovery, [Online]. Saatavissa: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419?sc=internet/evalboard/product/252419.jsp# [Haettu 20.10.2015].

[4] J. Fonselius, E. Laitinen, K. Pekkola, A. Sampo, T. Välimaa, Koneautomaatio-anturit, Opetushallitus, 1988.

[5] P. Kuivalainen, Mikroanturit, Otatieto Oy, 1992.

[6] V.K. Varadan, V.V. Varadan, ”Microsensors, microelectromechanical systems(MEMS), and electronics for smart structures and systems”, *Smart matterials and structures*, osa/vuosik. 6, nro 9, 1999. [Online]. Saatavissa: http://iopscience.iop.org/article/10.1088/0964-1726/9/6/327/pdf [Haettu 25.10.2015].

[7] W. Kuehnel, Modelling of the mechanical behaviour of a differential capacitor acceleration sensor, *Sensors and Acuators A: Physical*, osa/vuosik. 48, nro 2, pp. 101–108, 1995. [Online]. Saatavissa: http://www.sciencedirect.com/science/article/pii/092442479400983O [Haettu 25.10.2015].

[8] J. Geier, Langattomat verkot, Edita Publishing Oy, 2004.

[9] K. Salkintzis, Interworking techniques and architectures for wlan/3G integration toward 4G mobile data networks, IEEE Wireless Communications, pp.50–61, 6/2004. [Online]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1308950 [Haettu 1.11.2015].

[10] K.R. Fall, W.R. Stevens, TCP/IP illustrated Volume 1- The protocols, Adison-Wesley, 2011.

[11] A.G. Blank, TCP/IP Jumpstart- Internet protocol basics, John Wiley & Sons, 2000.

[12] K. Kaario, TCP/IP-verkot, Docendo Finland Oy, 2002.

[13] STMicroelectronics, “STM32F407 dokumentaatio”, [Online]. Saatavissa: http://www.st.com/st-web-

ui/static/active/en/resource/technical/document/datasheet/DM00037051.pdf [Haettu 1.11.2015]

[14] Element14, STM32F407-Discovery-Wi-Fi user manual, [Online]. Saatavissa: http://www.element14.com/community/servlet/JiveServlet/previewBody/55281-102-2-287529/Discover%20Wi-Fi%20UserManual_V1.5.pdf [Haettu 4.11.2015].

[15] Silicon labs, UART-USB-bridge dokumentaatio, [Online]. Saatavissa: https://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf [Haettu 4.11.2015]

[16] Keil, MDK-ARM version 5 – Microcontroller development kit, [Online]. Saatavissa: http://www.keil.com/product/brochures/mdk.pdf. [Haettu 5.11.2015].

[17] S. Tilkov, S Vinoski, Node.js: Using JavaScript to Build High-Performance Network Programs, IEEE Internet computing, IEEE Computer Society, pp. 80–83, 2010. [Online]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5617064 [Haettu 5.11.2015].

[18] X. Gu, L. Yang, S. Wu, A real-time stream system based on node.js, pp. 479–482, 2014,[Online]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7073454 [Haettu 12.11.2015].

[19] Element14, Wi-Fi-ohjainpiirin ohjelmointikirjasto,[Online] Saatavissa: http://www.element14.com/community/community/designcenter/stm32f4-discovery-expansion-boards. [Haettu 12.11.2015].

[20] STMicroelectronics, LIS3DH-Anturin dokumentaatio, [Online]. Saatavissa: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00274221.pdf [Haettu 15.11.2015].

[21] Node.js, Node.js dokumentaatio, [Online]. Saatavissa: https://nodejs.org/en/ [Haettu 15.11.2015].

[22] Mathworks, Simulink dokumentaatio, [Online]. Saatavissa http://se.mathworks.com/products/simulink/ [Haettu 15.11.2015].

[23] Aimagin, Waijung blockset dokumentaatio, [Online]. Satavissa: http://waijung.aimagin.com/ [Haettu 15.11.2015]

LIITE A: C++ -KOODI

Liitetiedosto sisältää mikrokontrolleriin ladattavan ohjelman osan. Tiedoston nimi on main.c.

```
/******************** (C) COPYRIGHT 2009 Embest Info&Tech Co.,LTD. ************
* File Name          : main.c
* Author             : Wuhan R&D Center, Embest
* Date First Issued  : 28/03/2013
* Description        : Main program body
********************************************************************************
********************************************************************************
* History:
* 28/03/2013     : V1      initial version
* 13/06/2013     : V2
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "sn8200_api.h"
#include "sn8200_core.h"
#include "delay.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define DBGU_RX_BUFFER_SIZE 256
#define TEST_BUFFERSIZE 128
#define UDP_NUM_PKT 10

/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
uint8_t DBGU_RxBuffer[DBGU_RX_BUFFER_SIZE];
uint32_t DBGU_RxBufferTail = 0;
uint32_t DBGU_RxBufferHead = 0;
bool DBGU_InputReady = false;
bool quit_flag = false;

uint8_t key;
uint8_t seqNo = 0;

int8_t mysock = -1;
int8u TxBuf[TEST_BUFFERSIZE];

extern int ipok, joinok;
extern int destIP, srcIP;
extern long int destPort, srcPort;
extern int32u pktcnt;

extern char domain[100];
extern char Portstr[8];
char uri[100]={0};
char sockConnected = -1;
char sockClosed = -1;
```

```
40  int8_t mysock = -1;
41  int8u TxBuf[TEST_BUFFERSIZE];
42
43  extern int ipok, joinok;
44  extern int destIP, srcIP;
45  extern long int destPort, srcPort;
46  extern int32u pktcnt;
47
48  extern char domain[100];
49  extern char Portstr[8];
50  char uri[100]={0};
51  char sockConnected = -1;
52  char sockClosed = -1;
53  int timeout1 = 5;
54  extern bool IsCreateSocketResponsed ;
55  extern int32u timeout;
56  extern bool IsWIFIJoinResponsed ;
57
58  //itse lisatyt muuttujat
59  char sockstrOMA[8];
60  int32u sockOMA;
61  char teststrOMA[128];
62  int lenOMA;
63  int8u a=0;
64
65  //Wifi asetukset
66  char WiFi_nimi[20]= "AirLink3G";
67  char turva[2]="4"; //Enter Security Mode (e.g., 0 for open, 2 for WPA TKIP, 4 for WPA2
68  char salasana[20]= "uJFwfebF";
69
70  char IPosoite[20] = "192.168.0.104";
71  int kohdeportti1 = 9001;
72  int kohdeportti2 = 9002;
73  int kohdeportti3 = 9003;
74
75  int8_t kiihtyvyys1;
76  int8_t kiihtyvyys2;
77  int8_t kiihtyvyys3;
78
79  int8_t mysock1 = -1;
80  int8_t mysock2 = -1;
81  int8_t mysock3 = -1;
82
83  SPI_InitTypeDef SPI_InitTypeDefStruct; // spi init nostettu ylos
84  GPIO_InitTypeDef GPIO_InitTypeDefStruct; //spi GPIO init
85  GPIO_InitTypeDef GPIO_InitDef; //led
86
87  bool lahetys = false;
88
89
90  #define GET_REQUEST \
91      "GET / HTTP/1.1\r\n" \
```

```
 82
 83  SPI_InitTypeDef SPI_InitTypeDefStruct; // spi init nostettu ylos
 84  GPIO_InitTypeDef GPIO_InitTypeDefStruct; //spi GPIO init
 85  GPIO_InitTypeDef GPIO_InitDef; //led
 86
 87  bool lahetys = false;
 88
 89
 90  #define GET_REQUEST \
 91      "GET / HTTP/1.1\r\n" \
 92      "Host: 192.168.2.125\r\n" \
 93      "Accept: text/html\r\n" \
 94      "\r\n"
 95
 96  /* Private function prototypes -----------------------------------------------*/
 97  #ifdef __GNUC__
 98  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
 99     set to 'Yes') calls __io_putchar() */
100  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
101  #else
102  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
103  #endif /* __GNUC__ */
104
105  void DBGU_Init(void);
106  bool DBGU_RxBufferEmpty(void);
107  uint8_t DBGU_GetChar(void);
108  void ShowMenu(void);
109  void ProcessUserInput(void);
110  int sendHttpReqTest(char *domain, char isHttps);
111  int sendHttpPostDemo(char *domain);
112  int sendHttpJsonPostDemo(char *domain);
113  int sendHttpChunkReqTest(char *domain);
114
115  /* Private functions ---------------------------------------------------------*/
116
117  /**
118    * @brief  Main program.
119    * @param  None
120    * @retval None
121    */
122
123  void MY_LEDinit(void){
124   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
125
126    GPIO_InitDef.GPIO_Pin = GPIO_Pin_0;
127    GPIO_InitDef.GPIO_Mode = GPIO_Mode_IN;
128    GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
129    GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_NOPULL;
130    GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
131    //Initialize pins
132    GPIO_Init(GPIOA, &GPIO_InitDef);
133  }
```

```
124   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
125
126    GPIO_InitDef.GPIO_Pin = GPIO_Pin_0;
127    GPIO_InitDef.GPIO_Mode = GPIO_Mode_IN;
128    GPIO_InitDef.GPIO_OType = GPIO_OType_PP;
129    GPIO_InitDef.GPIO_PuPd = GPIO_PuPd_NOPULL;
130    GPIO_InitDef.GPIO_Speed = GPIO_Speed_100MHz;
131    //Initialize pins
132    GPIO_Init(GPIOA, &GPIO_InitDef);
133  }
134   //kiihtyvyysanturin alustus ja datanhakufunktio
135  void mySPI_Init(void){
136
137
138    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
139
140
141    SPI_InitTypeDefStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
142    SPI_InitTypeDefStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
143    SPI_InitTypeDefStruct.SPI_Mode = SPI_Mode_Master;
144    SPI_InitTypeDefStruct.SPI_DataSize = SPI_DataSize_8b;
145    SPI_InitTypeDefStruct.SPI_NSS = SPI_NSS_Soft;
146    SPI_InitTypeDefStruct.SPI_FirstBit = SPI_FirstBit_MSB;
147    SPI_InitTypeDefStruct.SPI_CPOL = SPI_CPOL_High;
148    SPI_InitTypeDefStruct.SPI_CPHA = SPI_CPHA_2Edge;
149
150    SPI_Init(SPI1, &SPI_InitTypeDefStruct);
151
152    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE , ENABLE);
153
154  //anturi on kiinni pinneissa A5-7
155    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7 | GPIO_Pin_6;
156    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_AF;
157    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
158    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
159    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
160    GPIO_Init(GPIOA, &GPIO_InitTypeDefStruct);
161
162    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_3;
163    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_OUT;
164    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
165    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_UP;
166    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
167    GPIO_Init(GPIOE, &GPIO_InitTypeDefStruct);
168
169    GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
170    GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
171    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
172
173    GPIO_SetBits(GPIOE, GPIO_Pin_3);
174
175
```

```
166     GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
167     GPIO_Init(GPIOE, &GPIO_InitTypeDefStruct);
168
169     GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
170     GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
171     GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
172
173     GPIO_SetBits(GPIOE, GPIO_Pin_3);
174
175
176     SPI_Cmd(SPI1, ENABLE);
177
178   }
179
180   uint8_t mySPI_GetData(uint8_t adress){
181
182     GPIO_ResetBits(GPIOE, GPIO_Pin_3);
183
184     adress = 0x80 | adress;
185
186     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE));  //transmit buffer empty?
187     SPI_I2S_SendData(SPI1, adress);
188     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE)); //data received?
189     SPI_I2S_ReceiveData(SPI1);  //Clear RXNE bit
190
191     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE));  //transmit buffer empty?
192     SPI_I2S_SendData(SPI1, 0x00); //Dummy byte to generate clock
193     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE)); //data received?
194     GPIO_SetBits(GPIOE, GPIO_Pin_3);
195
196     return SPI_I2S_ReceiveData(SPI1); //return reveiced data
197   }
198   void mySPI_SendData(uint8_t adress, uint8_t data){
199
200     GPIO_ResetBits(GPIOE, GPIO_Pin_3);
201
202     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE));  //transmit buffer empty?
203     SPI_I2S_SendData(SPI1, adress);
204     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE)); //data received?
205     SPI_I2S_ReceiveData(SPI1);
206
207     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE));  //transmit buffer empty?
208     SPI_I2S_SendData(SPI1, data);
209     while(!SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE)); //data received?
210     SPI_I2S_ReceiveData(SPI1);
211
212     GPIO_SetBits(GPIOE, GPIO_Pin_3);
213   }
214
215
216
217   int main(void)
```

```
211
212     GPIO_SetBits(GPIOE, GPIO_Pin_3);
213   }
214
215
216
217   int main(void)
218   {
219       SysTick_Configuration();
220       DBGU_Init();
221       SN8200_API_Init(921600);
222       strcpy(domain, "www.murata-ws.com");
223       strcpy(uri, "/index.html");
224
225       printf("\n\rHello, Embedded World!\n\r");
226
227       printf("\n\r");
228
229       WifiOn(seqNo++);
230       printf("\n\r");
231
232     //luodaan yhteys muokatuilla funktioilla, lisataan myohemmin syoteena SSID secutaso ja salasanasyote
233           WifiDisconn(seqNo++);
234           WifiJoinOMA(seqNo++,WiFi_nimi,turva,salasana);
235           SnicInit(seqNo++);
236           SnicIPConfig(seqNo++);
237
238     //luodaan ensimainen socketti palvelimelle
239     mysock = -1;
240     tcpCreateSocket(0, 0xFF, 0xFF, seqNo++, SNIC_TCP_CREATE_SOCKET_REQ); //sni_tcp:sta kalastellaan oman
241           if (mysock != -1) {
242             printf("mysokin arvo: %d",mysock);
243             mysock1 = mysock;
244               if (getTCPinfoOMA(kohdeportti1,IPosoite) == CMD_ERROR) {
245                   printf("Invalid Server\n\r");
246
247               }
248               // This connection can receive data upto 0x0400=1K bytes at a time.
249               printf("luotiinko jo soketti?");
250               tcpConnectToServer(mysock1, destIP, (unsigned short)destPort, 0x0400, 0x5, seqNo++);
251           }
252     //luodaan toinen socketti palvelimelle
253
254     mysock = -1;
255     tcpCreateSocket(0, 0xFF, 0xFF, seqNo++, SNIC_TCP_CREATE_SOCKET_REQ); //sni_tcp:sta kalastellaan oman
256           if (mysock != -1) {
257             mysock2 = mysock;
258             printf("mysokin arvo: %d",mysock);
259               if (getTCPinfoOMA(kohdeportti2,IPosoite) == CMD_ERROR) {
260                   printf("Invalid Server\n\r");
261
262               }
```

```
250             tcpConnectToServer(mysock1, destIP, (unsigned short)destPort, 0x0400, 0x5, seqNo++);
251         }
252     //luodaan toinen socketti palvelimelle
253
254     mysock = -1;
255     tcpCreateSocket(0, 0xFF, 0xFF, seqNo++, SNIC_TCP_CREATE_SOCKET_REQ); //sni_tcp:sta kalastellaan oman
256         if (mysock != -1) {
257           mysock2 = mysock;
258           printf("mysokin arvo: %d",mysock);
259             if (getTCPinfoOMA(kohdeportti2,IPosoite) == CMD_ERROR) {
260                 printf("Invalid Server\n\r");
261
262             }
263             // This connection can receive data upto 0x0400=1K bytes at a time.
264             printf("luotiinko jo soketti?");
265             tcpConnectToServer(mysock2, destIP, (unsigned short)destPort, 0x0400, 0x5, seqNo++);
266         }
267
268       mysock = -1;
269     tcpCreateSocket(0, 0xFF, 0xFF, seqNo++, SNIC_TCP_CREATE_SOCKET_REQ); //sni_tcp:sta kalastellaan oman
270         if (mysock != -1) {
271           printf("mysokin arvo: %d",mysock);
272           mysock3 = mysock;
273           if (getTCPinfoOMA(kohdeportti3,IPosoite) == CMD_ERROR) {
274                 printf("Invalid Server\n\r");
275
276             }
277             // This connection can receive data upto 0x0400=1K bytes at a time.
278             printf("luotiinko jo soketti?");
279             tcpConnectToServer(mysock, destIP, (unsigned short)destPort, 0x0400, 0x5, seqNo++);
280         }
281
282         //alustetaan SPI väylä
283         mySPI_Init();
284         MY_LEDinit(); //alustetaan kayttajan button
285
286         mySPI_SendData(0x20, 0x67); //asetetaan ctrlreg_4 kaikki axelit kaytettavaksi
287         mySPI_SendData(0x24, 0x00); //asetetaan tarkkuus 2G/4G/8Gjne
288
289       ShowMenu();
290
291       /* Infinite loop */
292       while (1) {
293         if(DBGU_InputReady) {
294           ProcessUserInput();
295         }
296
297         if(SN8200_API_HasInput()) {
298             ProcessSN8200Input();
299         }
300         if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)){
301           lahetys = true;
```

```
286         mySPI_SendData(0x20, 0x67); //asetetaan ctrlreg_4 kaikki axelit kaytettavaksi
287         mySPI_SendData(0x24, 0x00); //asetetaan tarkkuus 2G/4G/8Gjne
288
289       ShowMenu();
290
291       /* Infinite loop */
292       while (1) {
293         if(DBGU_InputReady) {
294           ProcessUserInput();
295         }
296
297         if(SN8200_API_HasInput()) {
298             ProcessSN8200Input();
299         }
300         if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)){
301           lahetys = true;
302           mdelay(100);
303         }
304
305         while(lahetys == true){
306
307           if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)){
308             lahetys = false;
309             mdelay(500);
310           }
311
312           kiihtyvyys1 = mySPI_GetData(0x29);    //poimitaan kiihtyvyysdata XYZ
313           kiihtyvyys2 = mySPI_GetData(0x2B);
314           kiihtyvyys3 = mySPI_GetData(0x2D);
315
316
317           //datan lahteys tcp porttiin
318           //tassa osassa kaytetyt muuttujat esitelty ylhaalla
319           snprintf(teststrOMA, sizeof(teststrOMA), "%d", kiihtyvyys1);     //muuttaa int tyyppisen
320           printf("anturi antoi1: %d ",kiihtyvyys1);
321           lenOMA = (int)strlen(teststrOMA);
322           sendFromSock(mysock1, (int8u*)teststrOMA, lenOMA, 2, seqNo++);
323
324           snprintf(teststrOMA, sizeof(teststrOMA), "%d", kiihtyvyys2);     //muuttaa int tyyppisen
325           printf("anturi antoi2: %d ",kiihtyvyys2);
326           lenOMA = (int)strlen(teststrOMA);
327           sendFromSock(mysock2, (int8u*)teststrOMA, lenOMA, 2, seqNo++);
328
329           snprintf(teststrOMA, sizeof(teststrOMA), "%d", kiihtyvyys3);     //muuttaa int tyyppisen
330           printf("anturi antoi3: %d ",kiihtyvyys3);
331           lenOMA = (int)strlen(teststrOMA);
332           sendFromSock(mysock3, (int8u*)teststrOMA, lenOMA, 2, seqNo++);
333
334           mdelay(300); //Delay lahetettavien pakettien valilla millisekunteina
335
336         }
337
```

```
325         printf("anturi antoi2: %d ",kiihtyvyys2);
326         lenOMA = (int)strlen(teststrOMA);
327         sendFromSock(mysock2, (int8u*)teststrOMA, lenOMA, 2, seqNo++);
328 
329         snprintf(teststrOMA, sizeof(teststrOMA), "%d", kiihtyvyys3);     //muuttaa int tyyppisen
330         printf("anturi antoi3: %d ",kiihtyvyys3);
331         lenOMA = (int)strlen(teststrOMA);
332         sendFromSock(mysock3, (int8u*)teststrOMA, lenOMA, 2, seqNo++);
333 
334         mdelay(300); //Delay lahetettavien pakettien valilla millisekunteina
335 
336         }
337 
338 
339         if(quit_flag)
340             break;
341     }
342 
343     printf("\n\rGoodbye, Embedded World!\n\r");
344 }
345 
346 void DBGU_Init(void)
347 {
404 bool DBGU_RxBufferEmpty(void)
405 {
410 void USART3_IRQHandler(void)
411 {
437 /**
442 PUTCHAR PROTOTYPE {
454 uint8_t DBGU_GetChar(void)
455 {
469 int fgetc(FILE *f)
470 {
487 void ShowMenu(void)
488 {
515 void ProcessUserInput(void)
516 {
750 #ifdef  USE_FULL_ASSERT
770 /**
773 int sendHttpReqTest(char *domain, char isHttps)
774 {
790 /**
794 int sendHttpPostDemo(char *domain)
795 {
815 /**
818 int sendHttpJsonPostDemo(char *domain)
819 {
841 /**
844 int sendHttpChunkReqTest(char *domain)
845 {
863 /************* (C) COPYRIGHT 2013 Wuhan R&D Center, Embest *****END OF FILE****/
864 
```