SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

RF-MIKROKONTROLLERI-KORTIN OHJELMOINTI

TEKIJÄ: Juho Hirvonen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Sähkötekniikan koulutusohjelma	
Työn tekijä(t) Juho Hirvonen	
Työn nimi RF-mikrokontrollerikortin ohjelmointi	
Päiväys 3.3.2017	Sivumäärä/Liitteet 28/10
Ohjaaja(t) yliopettaja Väinö Maksimainen, osakas Sami Kantoluoto, osakas Jukka Marin	
Toimeksiantaja/Yhteistyökumppani(t) Embedtronics Oy	

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli tehdä Texas Instrumentsin CC1200-radiopiirin ajurit STMicroelectronicsin mikroprosessorille. CC1200-radiopiiri lähettää ja vastaanottaa dataa langattomasti radiotaajuuksien välityksellä ja sen toimintaa ohjataan mikrokontrollerilla SPI-väylän kautta. Työssä tehtiin myös ajurit kiihtyvyysanturille ja lämpötila-kosteusanturille. Jatkokäytön kannalta ajureiden oli oltava yksinkertaisia ja helppokäyttöisiä.

Työ toteutettiin kuopiolaiselle Embedtronics Oy:lle, joka suunnittelee ja tuottaa sulautettuja järjestelmiä ja niiden ohjelmistoja. Työn ohjelmoinnit suoritettiin tekstieditorilla Linux-ympäristössä ja koodit käännettiin GNU Compiler Collection -kääntäjällä. Debuggaus tapahtui UART-väylän kautta sarjaporttiin lähetettävillä teksteillä sekä kortin ledeillä. Oskilloskooppia ja yleismittaria käytettiin rautatason toiminnan varmistamisessa ja virheiden etsinnässä.

Opinnäytetyön tuloksena saatiin CC1200-radiopiirin C-kieliset ajurit, joilla onnistui piirin rekisterien kirjoitus ja luku sekä korttien välille saatiin muodostettua toimiva radioyhteys. Myös kiihtyvyysanturin sekä lämpötila-kosteusanturin ajurit onnistuivat, joskin näistä ei kaikkia ominaisuuksia tarkasteltu, koska työn varsinainen painopiste oli radiopiiri.

Avainsanat
Texas Instruments CC1200, STM32CubeMX, mikrokontrollerin ohjelmointi, GCC-kääntäjä

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electrical Engineering			
Author(s) Juho Hirvonen			
Title of Thesis Programming of RF-Microcontroller Board			
Date	3 March 2017	Pages/Appendices	28/10
Supervisor(s) Mr Väinö Maksimainen, Principal Lecturer, Mr Sami Kantoluoto, Shareholder, Mr Jukka Marin, Shareholder			
Client Organisation /Partners Embedtronics Oy			

Abstract

The purpose of this thesis was to create a driver for the Texas Instruments CC1200 transceiver to be used with STMicroelectronics's microcontroller. CC1200 transmits and receives data wirelessly through radio frequencies and it is controlled by microcontroller's SPI interface. In addition, the driver software for the acceleration sensor and the temperature-humidity sensor was created. For the further use of the drivers they had to be simple and easy to use.

The thesis was made for a local company called Embedtronics Oy. The company designs and produces embedded systems and software. The programming was carried out in the Linux environment with text editor and the sourcecode was compiled with GNU Compiler Collection. Text notifications through UART and onboard leds were used for debugging. An oscilloscope and multimeter were used for finding bugs and making sure that the hardware works properly.

As a result of this thesis, CC1200 radio chip driver software with the C language was made. With the driver software the radio registers can be written and read as well as RF connection was established between the boards. Also the driver software for the acceleration sensor and the temperature-humidity sensor succeeded although without all functions because the main focus was on the radio driver.

Keywords
Texas Instruments CC1200, STM32CubeMX, microcontroller programming, GNU Compiler Collection

ESIPUHE

Opinnäytetyöni tein Embedtronics Oy:lle kevään 2017 aikana. Työstä sain paljon kokemusta sulautettujen järjestelmien ohjelmoinnista, Linux-ympäristöstä, radiotiedonsiirrosta ja sen ohjelmoinnista. Kiinnostus sulautettuja järjestelmiä kohtaan kasvoi entisestään työn teon aikana. Uskon opinnäytetyötä tehdessä oppimieni asioiden auttavan minua työelämässä, kuin myös mahdollisissa omissa harrasteprojekteissani.

Haluan kiittää saamastani ohjauksesta, neuvoista ja opeista osakas Sami Kantoluotoa, osakas Jukka Marinia sekä yliopettaja Väinö Maksimaista.

Kuopiossa 20.4.2017
Juho Hirvonen

SISÄLTÖ

LYHENTEET JA KÄSITTEET

AD	Analog to Digital
APB	Advanced Peripheral Bus
CC1200	Texas Instrumentin valmistama matalavirtainen RF lähetin-vastaanotin
CMSIS	Cortex Microcontroller Software Interface Standard
CS	Chip Select
CRC	Cyclic Redundancy Check
DA	Digital to Analog
DC	Direct Current
eWOR	enchanced Wake On Radio
FIFO	First-in-First-Out
FSK	Frequency shift keying
GCC	GNU Compiler Collection
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
MCU	Microcontroller Unit
MSK	Minimum Shift Keying
OOK	On-Off Keying
RF	Radio Frequency
RISC	Reduced Instruction Set Computing
SPI	Serial Peripheral Interface

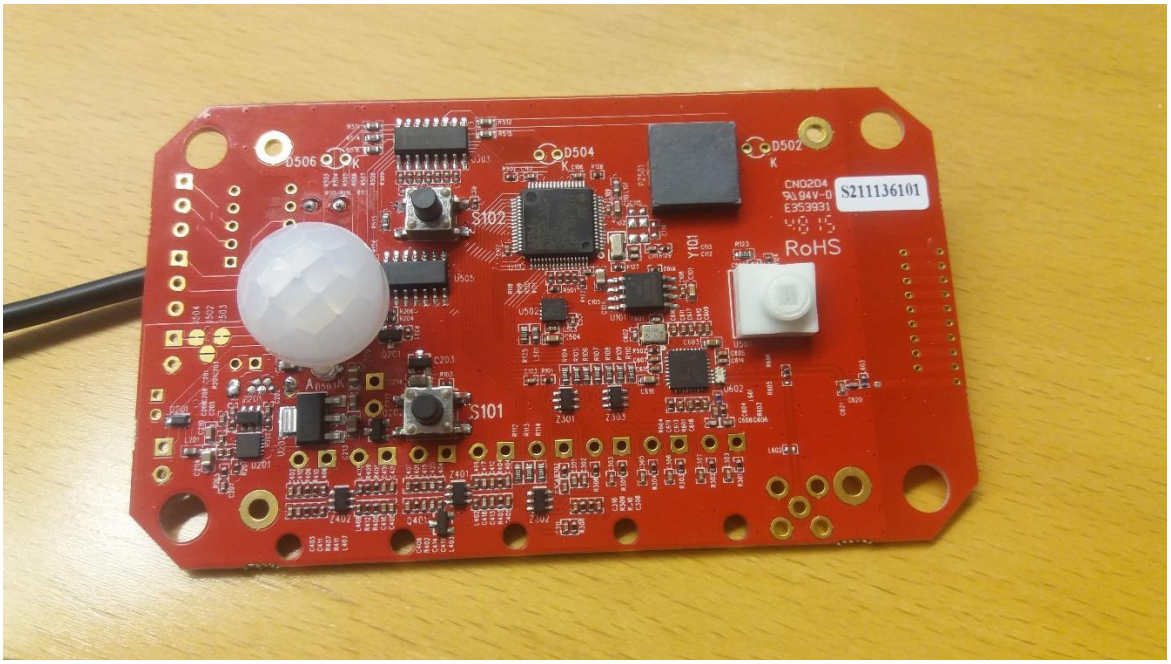
1 JOHDANTO

Opinnäytetyön RF-mikrokontrollerikortin ohjelmointi aihe saatiin kuopiolaiselta Embedtronics Oy:ltä, joka suunnittelee ja tuottaa sulautettujen järjestelmien ohjelmistoja ja laitteita. RF-taajuuksia on käytetty jo pitkään langattomaan tiedonsiirtoon, mutta nykypäivänä halutaan entistä matalavirtaisempia laitteita ja langaton tiedonsiirto on aina vienyt suuren osan laitteen käyttämästä virrasta. Embedtronicsin suunnittelema mikrokontrollerikortti (kuva 1) käyttää erittäin matalavirtaista radiopiiriä tiedonsiirtoon, joten laitteelle saadaan pitkä toiminta-aika paristoilla.

Työn tavoitteena on tehdä Texas Instrumentsin CC1200-radiopiirin sekä muiden kortilla olevien oheislaitteiden ajurit STMicroelectronicsin mikroprosessorille.

Työ toteutetaan Linux-ympäristössä käyttäen Pluma-tekstieditoria. Koodin kääntäjänä työssä käytetään GCC-kääntäjä, sekä mikroprosessorin ohjelmointivälineenä STM32flash. Mikrokontrolleria käytetään CMSIS-ohjelmistorajapinnalla ja STM32L1 HAL-ajureilla.

Työssä kerrotaan hyvin yleisesti työn suorittamisesta, eikä työssä paljasteta mikrokontrollerikortin yksityiskohtia tai koodeja salassapitosyistä.



KUVA 1. Työssä ohjelmoitava RF-mikrokontrollerikortti. (Hirvonen)

2 TEXAS INSTRUMENTS CC1200

Texas Instrumentsin CC1200 (kuva 2) on matalavirtainen ja -jännitteinen RF-lähetin-vastaanotin, joka on suunniteltu todella matalavirtaiseksi ja kustannustehokkaaksi ratkaisuksi langattomiin järjestelmiin. Piiriin on integroitu kaikki tarvittavat suodattimet, minkä vuoksi ei tarvitse lisätä kalliita ulkoisia suodattimia. CC1200 tarjoaa myös laajan tuen laitetasolla paketin käsittelyyn, datan bufferointiin, purskelähetyksiin, vapaan kanavan määritykseen, yhteyden laadun tarkkailuun ja Wake-On-Radio toimintoon. (Texas Instruments, 2013b)

KUVA 2. Texas Instrumentsin CC1200-radiopiiri (Texas Instruments, 2017)

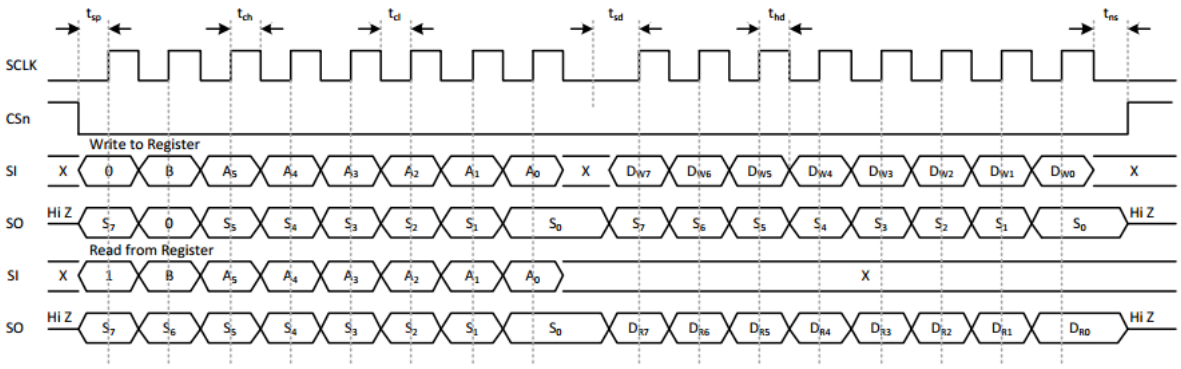
2.1 CC1200-radiopiirin käyttöliittymä

CC1200-radiopiiri on tarkoitettu käytettäväksi mikrokontrollerin kanssa ja yhteys näiden välillä tapahtuu 4-johdin SPI-väylällä, jossa CC1200 toimii slave-tilassa. CC1200-radiopiirissä on osoiteavaruus piirin asetuksia, sekä tulevaa ja lähtevää dataa varten.

2.1.1 4-johdin SPI-väylän käyttö

Normaalisti 4-johdin SPI-väylässä johtimet on nimetty MOSI, MISO, SCLK ja CS. MOSI-johtimessa näkyy datan tila mikrokontrollerilta radiopiirille ja MISO-johtimessa näkyy datan tila radiopiiriltä mikrokontrollerille. SCLK-johtimessa on mikrokontrollerin väylälle antama kellotaajuus, jonka tahtiin molempien datajohtimien tilojen vaihto tapahtuu. Samassa SPI-väylässä on usein monia oheislaitteita, kullekin laitteelle on oma CS-johtimensa. CS-johtimen avulla mikrokontrolleri kertoo, minkä oheislaitteen kanssa haluaa milloinkin muodostaa yhteyden.

CC1200-radiopiirin käyttöliittymä mikrokontrollerin kanssa on kohtuullisen yksinkertainen. Yhteysvaihtoehdot ovat rekisteritiedon kirjoitus ja luku, sekä näille molemmille purskemoodi. Purskemoodissa piirille voidaan kirjoittaa tai lukea useamman rekisterin tieto kerralla ilman, että tarvitsee aloittaa uutta yhteystapahtumaa. Kuviossa 1 on SPI-väylän ajoituskaavio.



KUVIO 1. Yhteys tapahtumien ajoituskaavio (Texas Instruments, 2013a)

Yhteystapahtuma alkaa, kun mikrokontrolleri vetää CS-linjan alas. Esimmäisen kellopulssin noustessa ylös CS-johtimen alasvedon jälkeen MOSI-johtimen tila kertoo radiopiirille, onko yhteydessä kyse rekisterin kirjoituksesta vai lukemisesta. Samalla radiopiiri lähettää MISO-pinnin tilalla mikrokontrollerille oman tilansa tietoa. Toisella pulssilla kerrotaan, onko yhteyden tarkoitus olla purskemoodissa. Nyt samalla pulssilla piiri lähettää toisen bitin tilansa tiedosta, mikäli kyse on lukutapahtumasta. Jos kyse on kirjoituksesta, niin piiri kirjoittaa tämän nollaksi ja tilatieto saadaan kokonaisuudessaa vasta datan kirjoitusvaiheessa. Kolmannella kellopulssilla aletaan antamaan käsiteltävän rekisterin osoitetta ja samalla radiopiiri antaa lopun tiedon tilastaan. Tätä jatkuu viiden pulssin ajan, minkä jälkeen piirille joko kirjoitetaan osoitteeseen haluttu data tai piiri kertoo osoitteessa olevan datan, mikäli kyse oli luku tapahtumasta. Yhteystapahtuman lopuksi mikrokontrolleri nostaa CS-linjan ylätilaan.

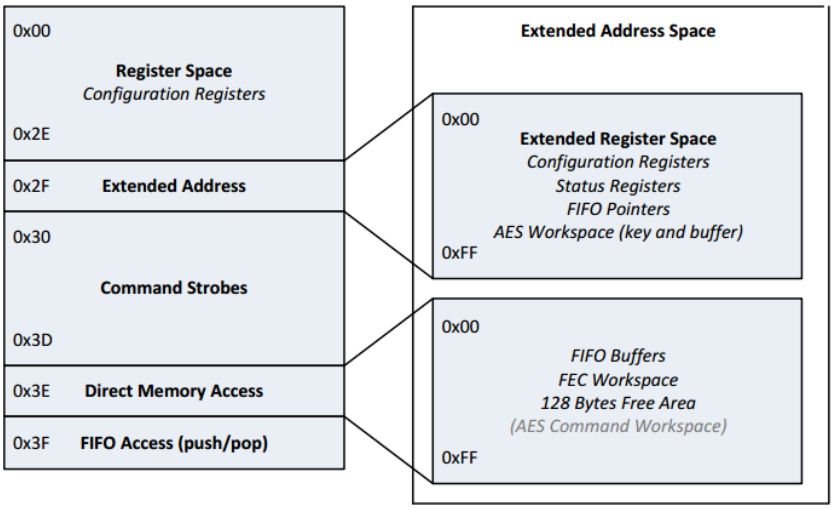
Poikkeuksia edeltäneisiin yhteystapahumiin ovat laajennetun rekisteriavaruuden luku ja kirjoitus, strobejen käyttö ja purskemoodi.

2.1.2 Purskemoodi

Purskemoodissa yhteystapahtuman aloitus on vastaava kuin yhden rekisteritiedon lukemisessa, mutta nyt mikrokontrollerilla ei nostetakaan CS-linjaa ensimmäisen datan kirjoituksen tai lukemisen jälkeen, vaan radiopiiri kasvattaa osoitetta yhdellä, ja seuraavalla kahdeksalla kellopulssilla luetaan tai kirjoitetaan osoitteeseen data. Tätä voidaan jatkaa niin kauan kuin halutaan ja lopuksi nostetaan CS-linja ylös. Mikäli radiopiirin osoitelaskuri saavuttaa arvon 0xFF, se pyörähtää ympäri arvoon 0x00. Purskemoodi on varsin kätevä piirin lähetys- ja vastaanottopuskurien kirjoitukseen ja lukuun.

2.1.3 Laajennetun rekisteriavaruuden käyttö

Kun halutaan käyttää laajennettua rekisteriavaruutta, annetaan ensimmäisenä osoitteen arvoksi 0x2F (kuva 3) ja heti seuraavilla kahdeksalla kellopulssilla annetaan käsiteltävä osoite laajennetusta rekisteriavaruudesta. Myös purskemoodi on käytettävissä laajennetun rekisteriavaruuden kanssa. (Texas Instruments, 2013a)



KUVA 3. Radiopiirin osoiteavaruudet. (Texas Instruments, 2013a)

2.1.4 Strobejen käyttö

CC1200-radiopiirille annetaan strobeja eli ”käskyjä”, joiden avulla käsketään radiopiiriä tekemään haluttu toiminto. Strobejen antaminen tapahtuu vastaavasti kuin rekisteriin kirjoitus, mutta annetaan vain halutun stroben osoite ja radiopiiri suorittaa sen välittömästi pois lukien strobet SRES, SPWD, SWOR ja SXOFF. Annettaessa normaali strobe voidaan sen jälkeen antaa välittömästi muita luku- tai kirjoituskäskyjä ilman, että CS-linjaa täytyy käyttää ylhäällä. SRES-stroben tilanteessa radiopiiri pitää SO-linjaa ylhäällä niin kauan, että piiri on valmis jatkamaan yhteyden pitoa (kuvio 2). Loppujen kolmen poikkeusstroben suoritus tapahtuu vasta, kun mikroprosessori nostaa CS-linjan. (Texas Instruments, 2013a)

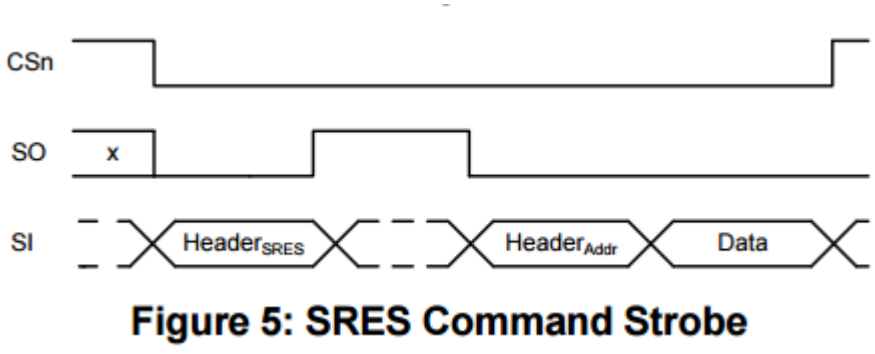


Figure 5: SRES Command Strobe

KUVIO 2. SRES-stroben ajoituskaavio. (Texas Instruments, 2013a)

Taulukosta 1 nähdään kaikkien mahdollisten strobejen osoitteet, nimet ja selitteet.

Address	Strobe Name	Description
0x30	SRES	Reset chip
0x31	SFSTXON	Enable and calibrate frequency synthesizer (if SETTLING_CFG.FS_AUTOCAL = 1). If in RX and PKT_CFG2.CCA_MODE ≠ 0: Go to a wait state where only the synthesizer is running (for quick RX/TX turnaround).
0x32	SXOFF	Enter XOFF state when CSn is de-asserted
0x33	SCAL	Calibrate frequency synthesizer and turn it off. SCAL can be strobed from IDLE mode without setting manual calibration mode (SETTLING_CFG.FS_AUTOCAL = 0)
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and SETTLING_CFG.FS_AUTOCAL = 1
0x35	STX	In IDLE state: Enable TX. Perform calibration first if SETTLING_CFG.FS_AUTOCAL = 1. If in RX state and PKT_CFG2.CCA_MODE ≠ 0: Only go to TX if channel is clear
0x36	SIDLE	Exit RX/TX, turn off frequency synthesizer and exit eWOR mode if applicable
0x37	SAFC	Automatic Frequency Compensation
0x38	SWOR	Start automatic RX polling sequence (eWOR) as described in Section 9.6 if WOR_CFG0.RC_PD = 0
0x39	SPWD	Enter SLEEP mode when CSn is de-asserted
0x3A	SFRX	Flush the RX FIFO. Only issue SFRX in IDLE or RX_FIFO_ERR states
0x3B	SFTX	Flush the TX FIFO. Only issue SFTX in IDLE or TX_FIFO_ERR states
0x3C	SWORRST	Reset the eWOR timer to the Event1 value
0x3D	SNOP	No operation. May be used to get access to the chip status byte

TAULUKKO 1. CC1200-radiopiirin command strobet. (Texas Instruments, 2013a)

2.2 CC1200:n konfigurointi

CC1200-radiopiirin rekisterien arvoja muuttamalla voidaan muokata piirin toimintaa hyvin monipuolisesti. Näistä tärkeimpiä ovat kantoaallon taajuus, modulaatioformaatti, lähetettävän paketin muoto, lähetysteho ja symbolitaajuus.

2.2.1 Kantoaallon taajuuden valinta

CC1200-radiopiiri on mahdollista konfiguroida käyttämään 164-190 MHz, 410-475 MHz ja 820-950 MHz taajuusalueita. Mikäli ryhtyy tekemään tuotetta, jossa käytetään radiotaajuuksia, tulee taajuusalueen luvanvaraisuus ja vaatimukset tarkastaa Viestintäviraston varattujen radiotaajuuksien taulukosta (Viestintävirasto, 2016).

2.2.2 Modulaatioformaatit

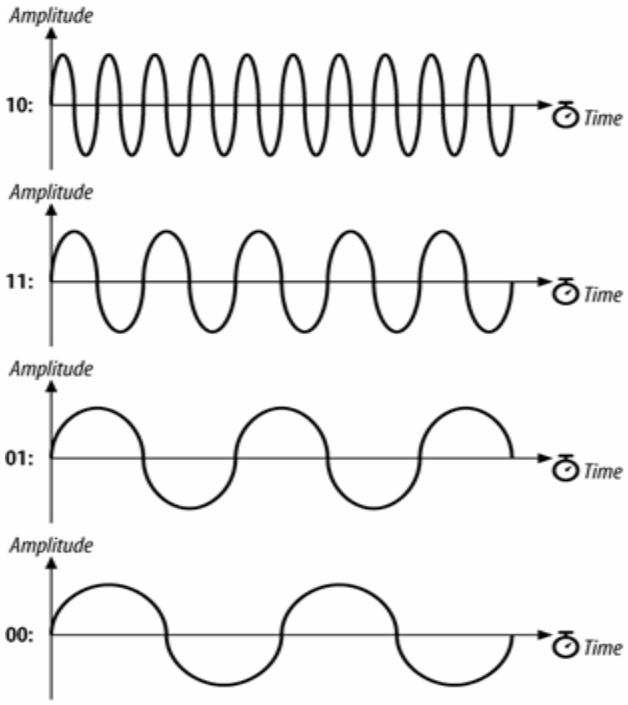
Radiopiiri tukee seuraavia modulaatio formaatteja

- 2-FSK
- 2-GFSK
- 4-FSK
- 4-GFSK
- MSK
- ASK
- OOK.

GFSK (Gaussian Frequency Shift Keying) ja FSK (Frequency Shift Keying) moduloivat datan sarjaksi taajuuden muutoksia kantoaallossa. Yksi taajuusmoduloidun datan eduista on, että sen kohina yleensä muuttaa signaalin amplitudia, joten taajuuden mukaan moduloitu signaali on suhteellisen immuuni kohinalle. (Gast, 2005)

FSK:n ja GFSK:n erona on, että GFSK:ssa signaalin reunat on pyöristetty, koska se ajetaan Gaussian suodattimen läpi. GFSK:n etuna on, että se käyttää vähemmän kaistanleveydestä ja näin on hiukan matalavirtaisempi kuin FSK, mutta FSK taas voi kuulua pidemmälle. (Gast, 2005)

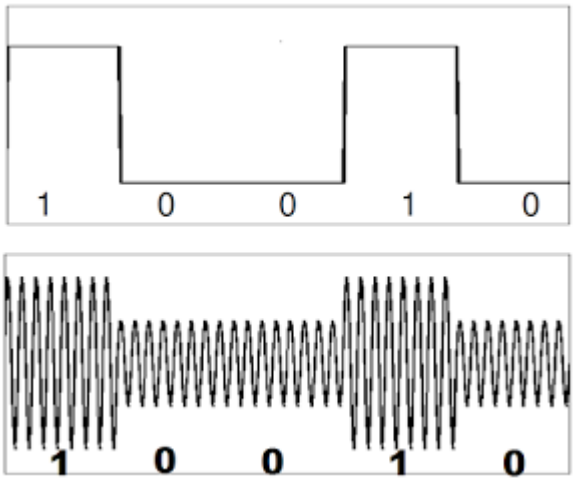
2-FSK ja 2-GFSK modulaatioilla saadaan yhdellä symbolilla annettua vain yksi databitti, kun taas 4-FSK:n ja 4-GSFK:lla saadaan yhteen symboliin kaksi bittiä (kuvio 3). 4-FSK:lla ja 4-GFSK:lla saadaan siis kaksi kertaa nopeampi datan siirto, mutta ne ovat taas herkempiä virheille, jonka takia kantama heikkenee. (Gast, 2005)



KUVIO 3. Kuvaus 4-GFSK symboleista. (Gast, 2005)

OOK ja ASK ovat signaalin amplitudin vaihteluun perustuvia modulaatiomuotoja. Amplitudimodulaation etuja ovat sen helppous, sekä pienempi kaistanleveyden tarve kuin FSK-modulaatioilla. Amplitudimodulaatiot ovat puolestaan herkempiä kohinan aiheuttamille virheille.

OOK moduloi datan kytkemällä lähetysvahvistimen päälle, kun data on 1 ja pois päältä, kun data on 0. ASK-modulaatio pienentää signaalin amplitudia datan ollessa 0 ja suurentaa datan ollessa 1 (kuvio 4).



KUVIO 4. ASK modulaation periaate. (Techiessms, 2016)

2.2.3 Paketin muoto

CC1200-radiopiirillä on sisäänrakennettu tuki pakettimuotoisille radioprotokollille. Radiopiirin tukemia paketin ominaisuuksia ovat (Texas Instruments, 2013b):

- purskeen alustustavut (Preamble bytes)
- synkronisaatiosana (Sync Word)
- CRC-laskenta ja tarkastus
- data whitening
- paketin pituuden määrittely
- laitteen osoite.

Purskeen alustustavujen tarkoitus on herättää vastaanottavan laite ja näin ilmoittaa, että lähetys on alkamassa. Purskeen alustustavuja on mahdollista valita neljästä erilaisesta, jotka ovat hexadesimaalisina lukuina 0xAA, 0x55, 0x33 ja 0xCC. Näiden lähetettävien alustustavujen määrä voidaan valita 0 ja 30 tavun väliltä. (Texas Instruments, 2013a)

Synkronisaatiosanan voi itse määritellä haluamakseen, mutta sen koko voi olla vain 11, 16, 18, 24 tai 32 bittiä. Synkronisaatiosanan tarkoitus on nimensä mukaisesti synkronisoida lähetys. Kun vastaanottava laite tunnistaa oikean synkronisaatiosanan, se aloittaa varsinaisen tiedon vastaanoton. (Texas Instruments, 2013a)

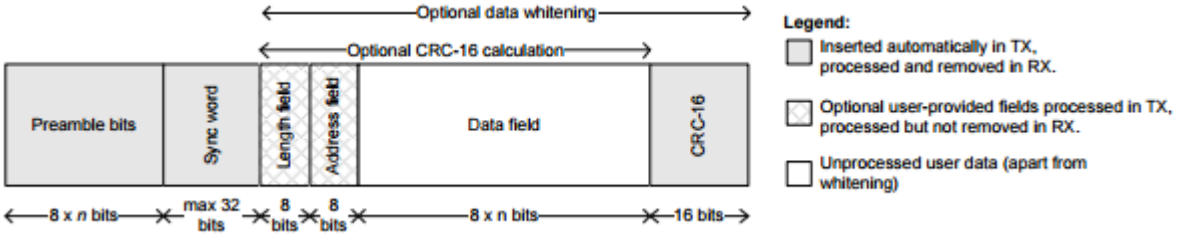
Radiopiiri lisää jokaiseen lähetykseen kaksi CRC-tavua, joilla voidaan tarkistaa lähetyksen eheys, mikäli käyttäjä näin haluaa. Radiopiiri voidaan asettaa automaattisesti suodattamaan virheelliset paketit CRC:n avulla tai antamaan mikrokontrollerille keskeytyssignaali, mikäli onnistunut paketti on vastaanotettu piirin FIFO-puskuriin. (Texas Instruments, 2013a)

Data whiteningin valinta onnistuu kirjoittamalla rekisteriin PKT_CFG1.WHITE_DATA arvon 1.

Radion perspektiivistä ideaali ilmateitse kulkeva data on satunnaista ja vapaa tasajännitteestä. Tämä johtaa tasaisimpaan tehonjakoon käytetyllä kaistanleveydellä. Todellinen data sisältää usein pitkiä jaksoja nollia ja ykkösiä tehden datan tunnistuksen hankalaksi. Näissä tapauksissa suorituskykyä voidaan parantaa käyttämällä data whiteningia ennen lähetystä ja purkamalla se vastaanotossa. (Texas Instruments, 2013b)

CC1200 tukee vakioituja ja muuttuvia paketin pituusprokollia. Molempia voidaan käyttää 255 tavuun asti. Pidempiä paketteja varten täytyy käyttää infinite packet length -moodia. Käytettäessä muuttuvia paketin pituuksia tulee paketin pituustieto heti synkronisaatiosanan jälkeen. (Texas Instruments, 2013a)

Pakettiin voidaan vielä lisätä osoitetieto pituustiedon jälkeen (kuvio 5), mikäli tälle on sovelluksessa tarvetta. Piirillä on sisäänrakennettu mahdollisuus suodattaa tulevat paketit osoitteen mukaan. Mikäli osoite ei ole oikea, piiri ei lisää pakettia FIFO-puskuriin vaan suodattaa sen. (Texas Instruments, 2013a)



KUVIO 5. Standardi paketin rakenne. (Texas Instruments, 2013a)

2.2.4 GPIO-ohjauspinnit mikrokontrollerille

CC1200-radiopiirillä on neljä digitaalista I/O pinniä GPIO0, GPIO1, GPIO2 ja GPIO3, jotka konfiguroidaan rekisterillä IOCFGx.GPIOx_CFG (jossa x on 0, 1, 2 tai 3). Liitteessä 2 on kaikki mahdolliset signaalit, joita voidaan seurata GPIO pinneillä. (Texas Instruments, 2013a)

GPIO1 on jaettu SPI-käyttöliittymän SO-linjan kanssa. Oletusasetus GPIO1/SO-pinnille on HIGHZ (kolmitaso) -ulostulo, joka on käytännöllinen, kun SPI-käyttöliittymä on jaettu muiden laitteiden kanssa. Valitsemalla jonkin muun asetuksen pinnille ohjelmointivaihtoehdoista, GPIO1/SO-pinnistä tulee generinen pinni CS-linjan ollessa ylätilassa ja SO-funktio sen ollessa alhaalla. (Texas Instruments, 2013a)

2.2.5 Enchanced Wake On Radio (eWOR)

Vaihtoehtoinen enchanced Wake On Radio (eWOR) -toiminto mahdollistaa CC1200:n asettamisen heräämään jaksottaisesti unitilasta ja kuuntelemaan tulevia paketteja ilman mikrokontrollerin vuorovaikutusta. Käyttämällä strobea SWOR CC1200 menee unitilaan CS-linjan vapautuessa. Radiopiirillä oleva eWOR-ajastin asettaa radiopiirin IDLE-tilan kautta RX-tilaan. Ohjelmoitavan ajan jälkeen piiri

menee takaisin SLEEP-tilaan, ellei pakettia otettu vastaan. Mikäli paketti otettiin vastaan, voidaan siitä ilmoittaa mikrokontrollerille jollain GPIO-linjoista. (Texas Instruments, 2013a)

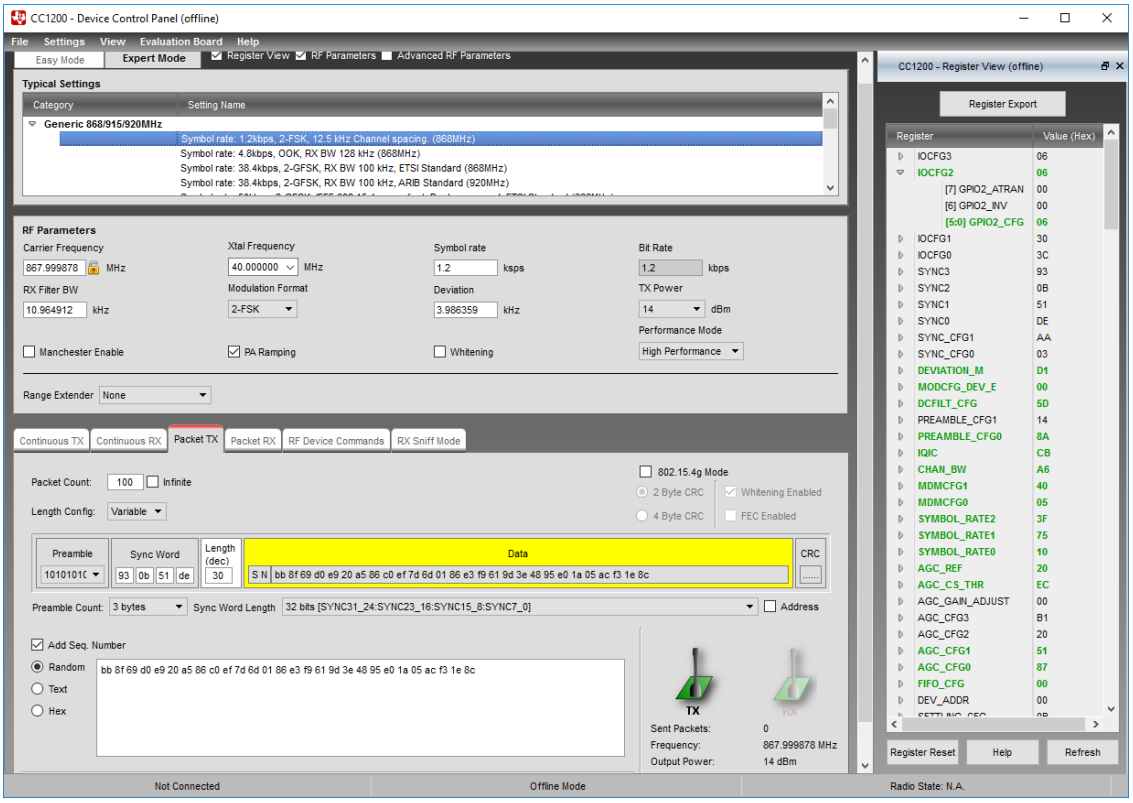
2.2.6 RX Sniff Mode

Paristoilla toimivissa järjestelmissä RX-teho on tärkeä parametri ja paristojen eliniän parantamiseksi uusi RX Sniff Mode -ominaisuus on suunniteltu CC120X-perheeseen haistelemaan autonomisesti RF-toimintaa käyttämällä ultra-low-power -algoritmia. CC1200 on suunniteltu todella nopealle asettumisajalle, joten vastaanotinta voidaan nopeasti kytkeä päälle ja pois päältä. Nopea ja varma RF-toiminnan tunnistus on avainparametri vähentämään virran kulutusta. RX Sniff Mode on erittäin hyödyllinen tapauksissa, mikäli ei tiedetä, milloin lähetin on lähettämässä pakettia. (Texas Instruments, 2013a)

2.3 SmartRF Studio 7

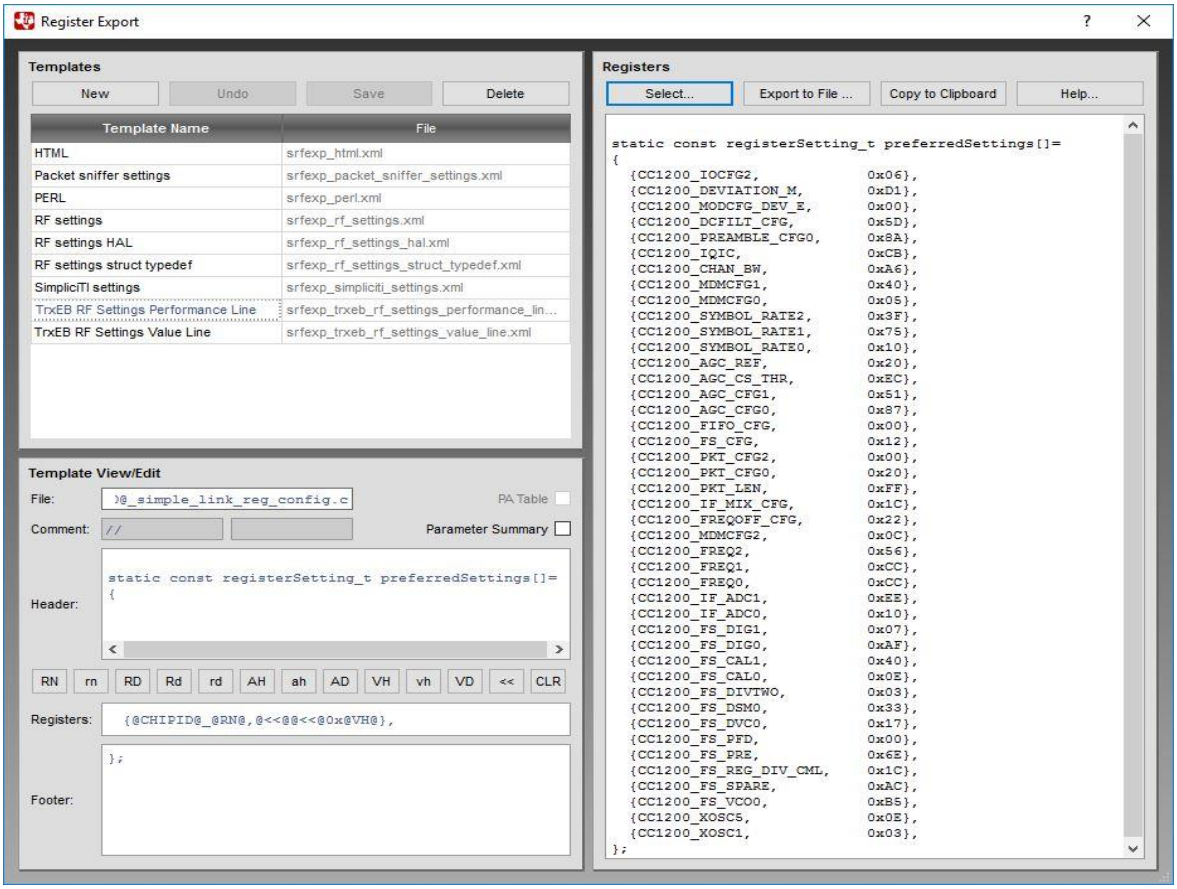
SmartRF Studio (kuva 4) on Texas Instrumentsin Windows sovellus, jolla voidaan konfiguroida ja testata erilaisia Texas Instrumentsin valmistamia RF-piirejä. Sovellus helpottaa RF-järjestelmien evaluointia kehitysprosessin alkuvaiheissa. Se on erityisen käytännöllinen rekisterien asetusarvojen valitsemiseen, testaamiseen käytännössä ja debuggaukseen. Sovellus tukee kaikkia Texas Instrumentsin matalavirtaisia RF-laitteita. (Texas Instruments, 2017)

Työssä käytettiin SmartRF Studioita eri asetusten kokeiluun. SmartRF:n ansiosta pystyi nopeasti kokeilemaan eri asetusarvoja ilman, että tarvitsi itse käydä kaikkia rekisterejä läpi ja laskea niihin tarvittavia arvoja halutuille asetuksille.



KUVA 4. SmartRF Studion näkymä asetus arvojen valinnassa.

SmartRF studiolla pystyy halutut asetusarvot tulostamaan kätevästi tiedostoon ja tekemään siitä headertiedoston projektiin (kuva 5). Radion konfigurointiheadertiedostoa vaihtamalla voidaan nopeasti testata radiopiirin toimivuutta eri asetuksilla. Tätä varten täytyi tehdä koodi, joka hakee alustusarvot tästä tiedostosta ja on yhteensopiva muuttujatyypin kanssa, eikä välitä montako rekisteriä kirjoitetaan.



KUVA 5. SmartRF Studion näkymä rekisteriarvojen viennistä tiedostoon.

3 STMICROELECTRONICS STM32L151-MIKROKONTROLLERI

ST:n ARM Cortex-M3:een perustuva STM32L1-sarja käyttää ST:n omistamaa ultra-low-leakage -prosessiteknologiaa innovatiivisen, autonomisen ja dynaamisen jänniteskaalauksen kanssa. Lisäksi viisi matalavirtaista moodia tarjoavat ennenkuulumattoman joustavan alustan sopimaan mihin tahansa sovellukseen. STM32L1-sarja laajentaa ultra-low-power -konseptia ilman kompromisseja suorituskyvyssä. (STMicroelectronics, STM32L1 Series, 2016b)

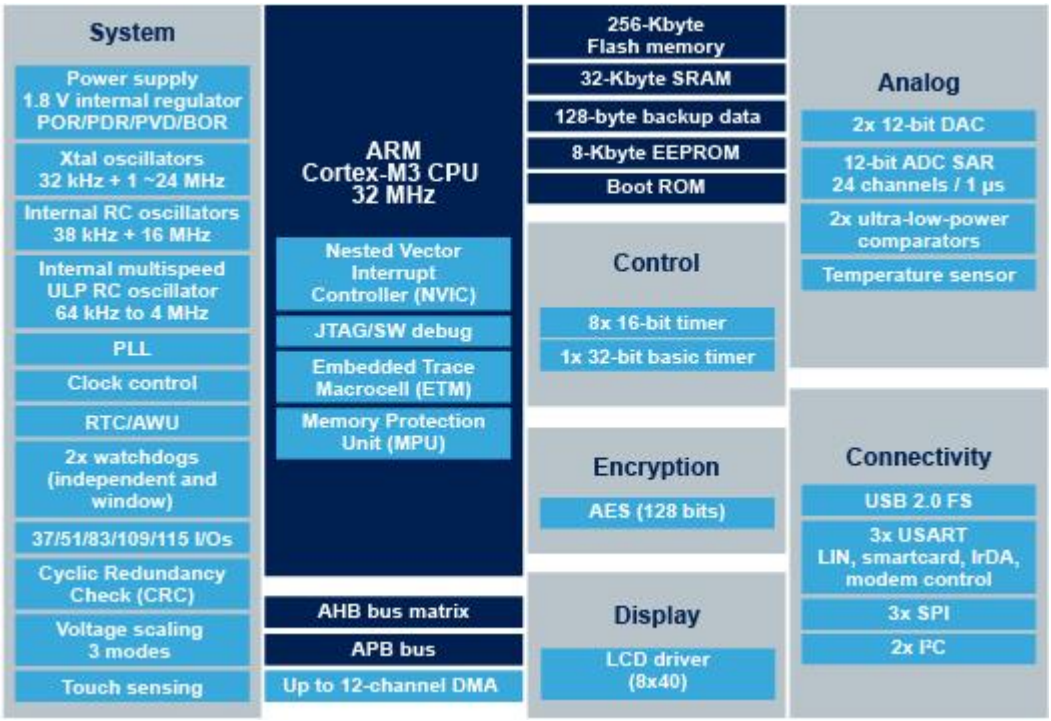
3.1 STM32L151RC:n ominaisuuksia

Ultra-low-power STM32L151xC-laitteet sisältävät USB-liitettävyyden korkea suorityskykyisen ARM Cortex-M3 32-bit 32 MHz RISC -ytimen kanssa, muistinsuojausyksikön, nopeat sulautetut muistit, laajennettu valikoima paranneltuja I/O-portteja ja oheislaitteet kytkettynä kahteen APB-väylään. (STMicroelectronics, STM32L151RC, 2016d)

STM32L151xC laitteet tarjoavat kaksi operaatiovahvistinta, yhden 12-bittisen AD-muuntimen, kaksi DA-muunninta, kaksi ultra-low-power -komparaattoria, yhden yleiskäyttöisen 32-bittisen ajastimen, kuusi yleiskäyttöistä 16-bittistä ajastinta ja kaksi normaalia ajastinta. Laitteet sisältävät myös seuraavat standardi ja kehittyneemmät kommunikointi käyttöliittymät: kaksi I2C-väylää, kolme SPI-väylää, kaksi I2S-väylää ja kolme USART-väylää sekä USB-väylän. (STMicroelectronics, STM32L151RC, 2017)

Avainominaisuudet:

- 1.65-3.6 V käyttöjännite
- 0.29 µA valmiustilassa kolmella herätyspinnillä
- 1.15 µA valmiustilassa reaaliaikakello päällä
- 8.6 µA matalavirta tilassa
- 185 µA/MHz normaalitilassa
- toiminta 32 kHz-32 MHz kellotaajuuksilla
- muistinsuojausyksikkö
- 256 KB Flash-muistia
- 32 KB RAM-muistia
- 8 KB EEPROM-muistia



KUVA 6. STM32L151RC-piirikaavio. (STMicroelectronics, STM32L151RC, 2016d)

3.2 CMSIS ja STM32L1 HAL

Työssä käytettiin CMSIS–ohjelmistorajapinnan CORE-osiota sekä STM32L1 HAL-laitteistoajureita.

CMSIS-CORE on valmistajasta riippumaton laitteistoabstraktiokerros Cortex-M prosessorisarjoille. Ohjelmiston tekeminen on suurin kulutekijä sulautettujen järjestelmien teollisuudessa. Ohjelmiston käyttöliittymän standardisointi kaikkien Cortex-M valmistajien välillä tarkoittaa merkittäviä säästöjä, erityisesti tehdessä uusia projekteja tai siirrettäessä olemassa olevaa ohjelmaa uuteen laitteeseen. (ARM, 2017)

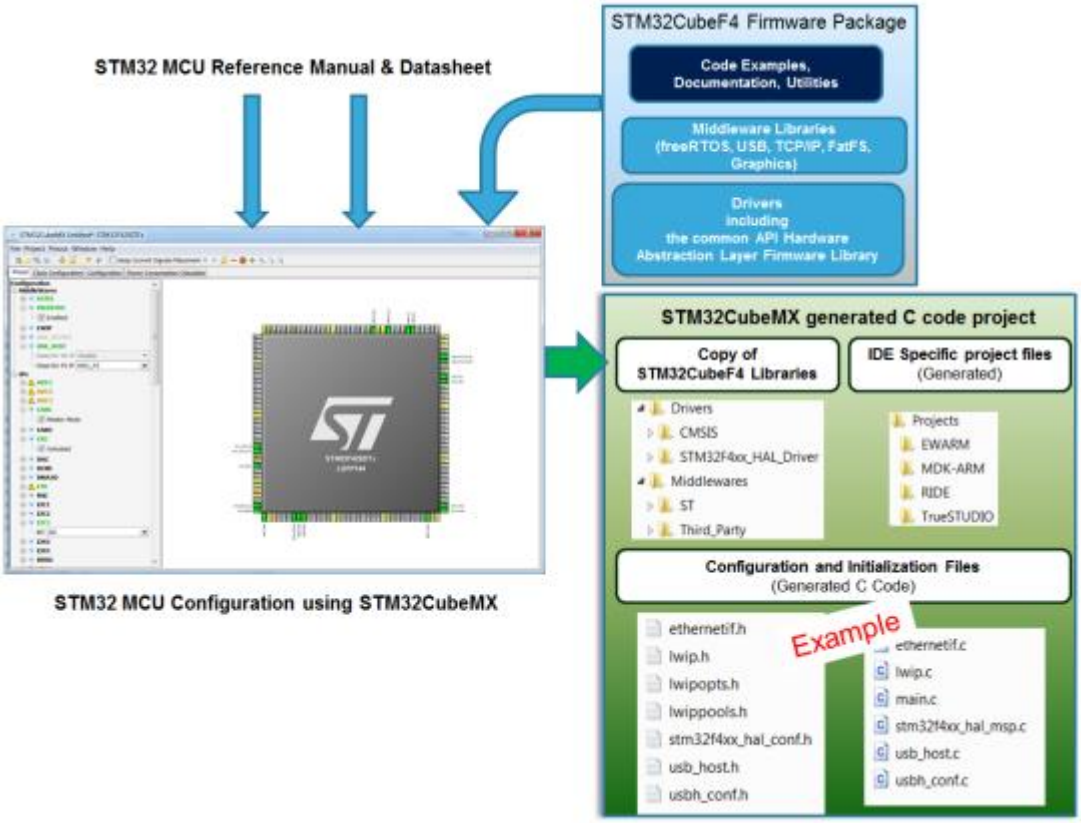
STM32L1 HAL-ajurit sisältävät täyden kokonaisuuden valmiita käyttöliittymiä, jotka yksinkertaistavat käyttäjän sovelluksen toteutusta. Esimerkiksi kommunikointioheislaitteiden käyttöliittymät sisältävät oheislaitteen alustuksen ja konfiguroinnin, datansiirron valinnan pollauksen, keskeytyksien ja suoran muistin käsittelyn välillä sekä virheiden käsittelyn. (STMicroelectronics, STM32L1 HAL, 2016c)

HAL-ajurit käyttävät CMSIS-ohjelmistorajapintaa, joten se on välttämätön, mikäli halutaan käyttää STMicroelectonicsin tarjoamia ajureita. Työssä tarvittiin joissain tilanteissa päästä käsiksi suoraan rekistereihin, joten CMSIS-rajapintaan täytyi tutustua tarkemmin.

3.3 STM32CubeMX

STM32CubeMX on osa sunniteltu helpottamaan kehittäjien elämää vähentämällä kehitystyötä, -aikaa ja -kustannuksia. STM32CubeMX on graafinen mikrokontrollerin konfigurointiväline, jolla voidaan generoida C-kielinen alustuskoodi (liite 4). (STMicroelectronics, STM32CubeMX, 2016a)

Sillä voidaan nopeasti ja helposti konfiguroida graafisesti mikrokontrollerille halutut toiminnot kuten pinnien ominaisuudet, kellon toiminta ja toimintamoodit. Haluttujen konfigurointien perusteella ohjelmalla voidaan tehdä alustuskoodi halutulle projektimuodolle. Ohjelma tukee EWARM, MDK-ARM TRUEStudio ja SW4STM32-kehitysympäristöjen projektimuotoja.



KUVA 7. STM32CubeMX-koodin generoinnin kulku (STMicroelectronics, STM32CubeMX, 2017)

3.4 STM32flash

STM32flash on avoimeen lähdekoodiin perustuva järjestelmäriippumaton flash-ohjelma. Ohjelma on tarkoitettu STM32 ARM-mikrokontrollereille, jotka käyttävät sisäänrakennettua ST serial bootloaderia UART:n tai I2C-väylän kautta. (stm32flash, 2016)

STM32flashin ominaisuuksia ovat:

- UART- ja I2C-väylien tuki
- laitteen tunnistaminen
- Flash- ja RAM-muistin luku ja kirjoitus
- binary-tiedostosta flashays
- muistin kirjoitusten verifiointi
- suorituksen aloitus halutusta osoitteesta
- laitteen resetointi ohjelmalla, kun kirjoitus on tehty.

4 TOTEUTUS

Työ tehtiin Linux-ympäristössä käyttäen Pluma-tekstieditoria ohjelmakoodin kirjoitukseen, GNU Compiler Collection -kääntäjää ohjelmakoodin kääntämiseen konekielelle, STM32CubeMX-ohjelmaa alustuskoodin luomiseen, sekä stm32flash-ohjelmaa mikrokontrollerin ohjelmointiin.

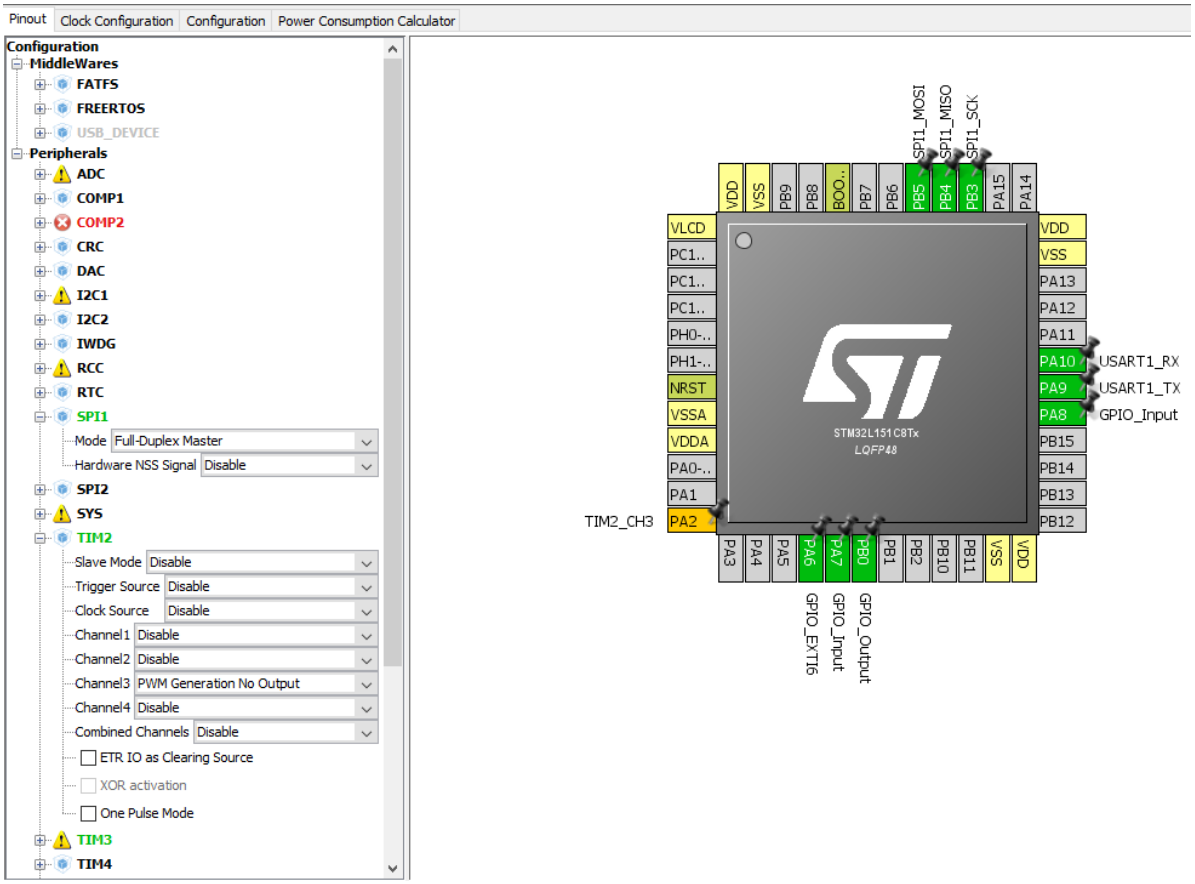
4.1 Työn aloitus

Työ aloitettiin perehtymällä Texas Instrumentsin CC1200-radiopiirin toimintaan ja varsinkin kommunikointiin mikrokontrollerin kanssa. Radiopiiristä kaikki tarpeellinen tieto työn suorittamista varten löytyi Texas Instrumentsin tarjoamasta piiriä koskevasta datalehdestä ja käyttäjäoppaasta.

Radiopiiriin perehtymisen jälkeen alettiin tutustua kortilla olleen STMicroelectronicsin mikrokontrollerin ominaisuuksiin ja tarjolla oleviin työkaluihin. Tähän löytyi hyvin kattavasti tietoa STMicroelectronicsin sivuilta ja hyviä esimerkkiohjelmia STM32L1 HAL -ajureiden käytöstä.

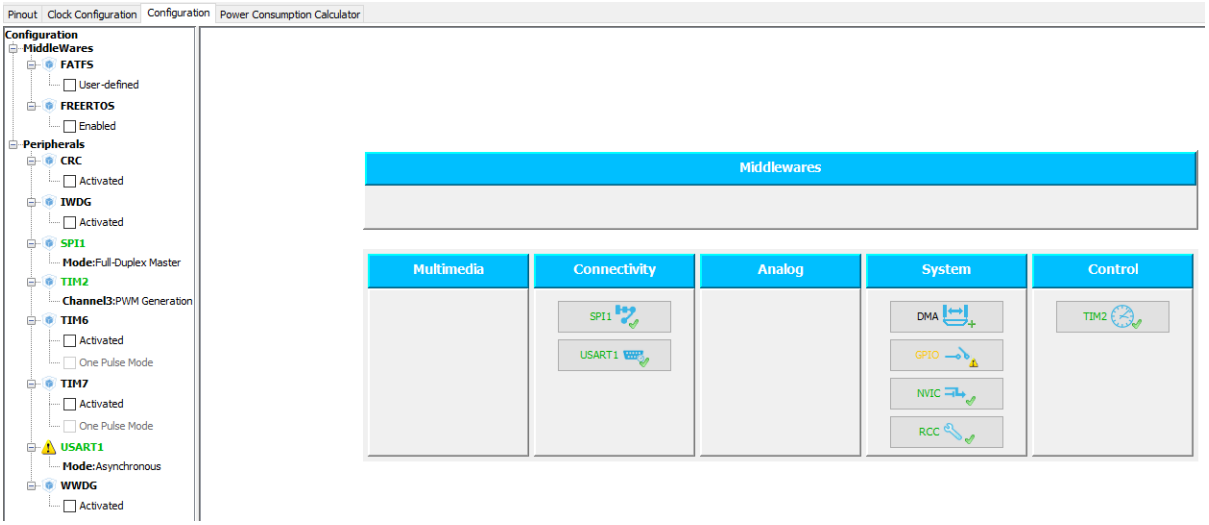
4.2 Alustuskoodin teko

Alustuskoodin tekoon käytettiin työssä STM32CubeMX-ohjelmaa. Ensimmäisenä ohjelmassa täytyy valita haluttu mikrokontrolleri, minkä jälkeen avautuu näkymä mikrokontrollerista (kuva 8). Tässä tilassa halutun pinnin kohdalta klikkaamalla sille voidaan antaa haluttu toiminto. Vasemmassa palkissa on asetuksia käytettävien oheislaitteiden toiminnoille. Seuraavana voitaisiin clock configuration välilehden alta asettaa kellolle graafisesti halutut asetukset, mutta tätä ei työssä tehty, koska se aiheutti tietokoneen jumiutumista Linux-ympäristössä. Kellon asetukset eivät olleet työssä kriittisiä. Lisäksi asetukset, joihin täytyi tehdä muutoksia, oli helppo itse muuttaa koodia muokkaamalla.



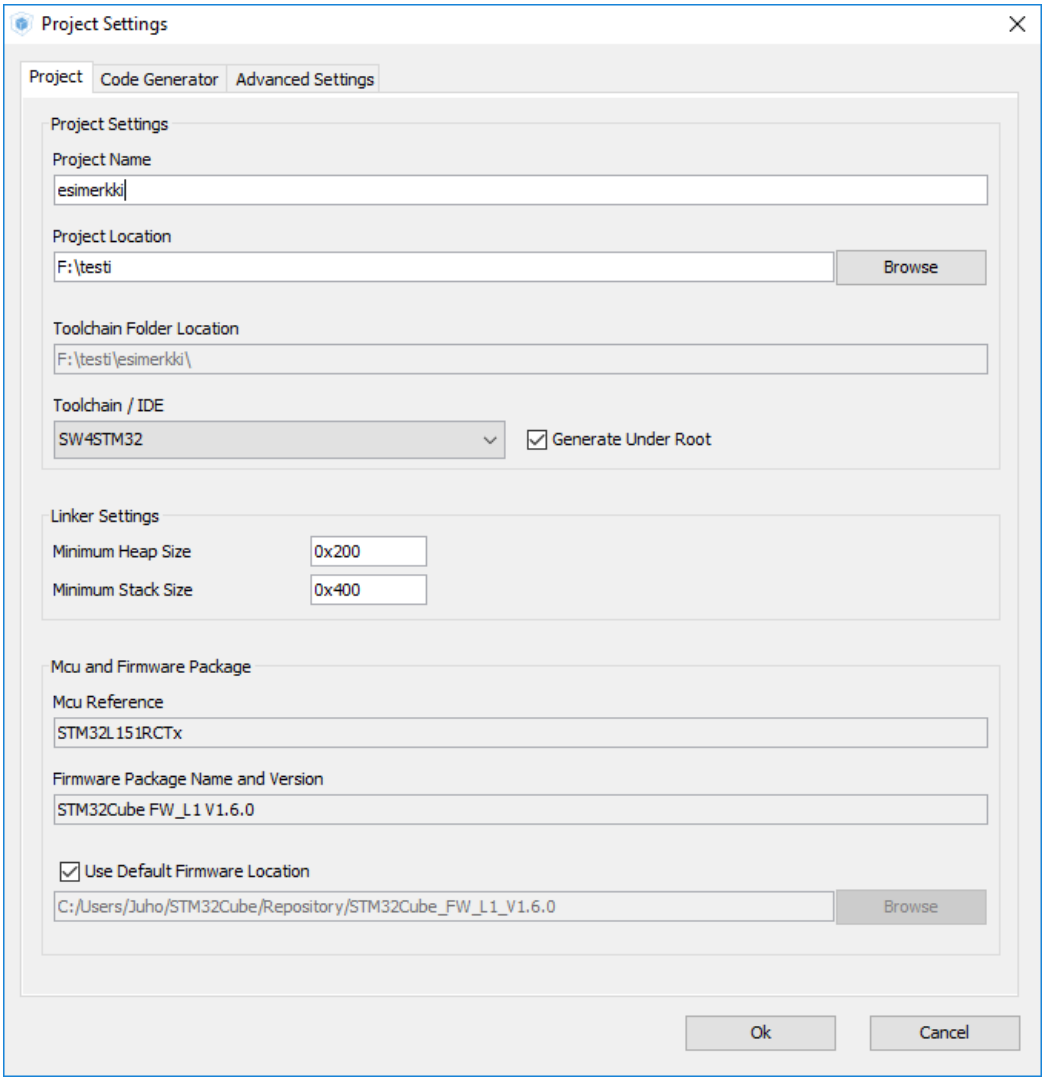
KUVA 8. STM32CubeMX-pinnien konfigurointinäkymä

Kun käytettävät pinnit saatiin valittua, voitiin oheislaitteiden ja keskeytyksien asetukset määrittää configuration-välilehdellä (kuva 9). Configuration-välilehdellä määritellään mm. SPI- ja UART -väylien asetukset, GPIO-pinnien toiminta, ajastinpiirin asetukset, laitteen keskeytykset ja niiden prioriteetit.



KUVA 9. STM32CubeMX oheislaitteiden konfigurointi välilehti.

Kun kaikki halutut konfiguroinnit saatiin tehtyä, oli jäljellä enää projektiin liittyvät asetukset. Projektin asetuksissa (kuva 10) määritellään projektin nimi, sijainti, kehitysympäristö, keko- ja pinomuistien minimikoko. Code Generator -välilehdellä on alustuskoodin luontiin liittyvät asetukset mm. säilytetäänkö käyttäjän kirjoittama koodi, asetetaanko käyttämättömät pinnit analogisiksi ja tehdäänkö varmuuskopiot edellisistä tiedostoista.

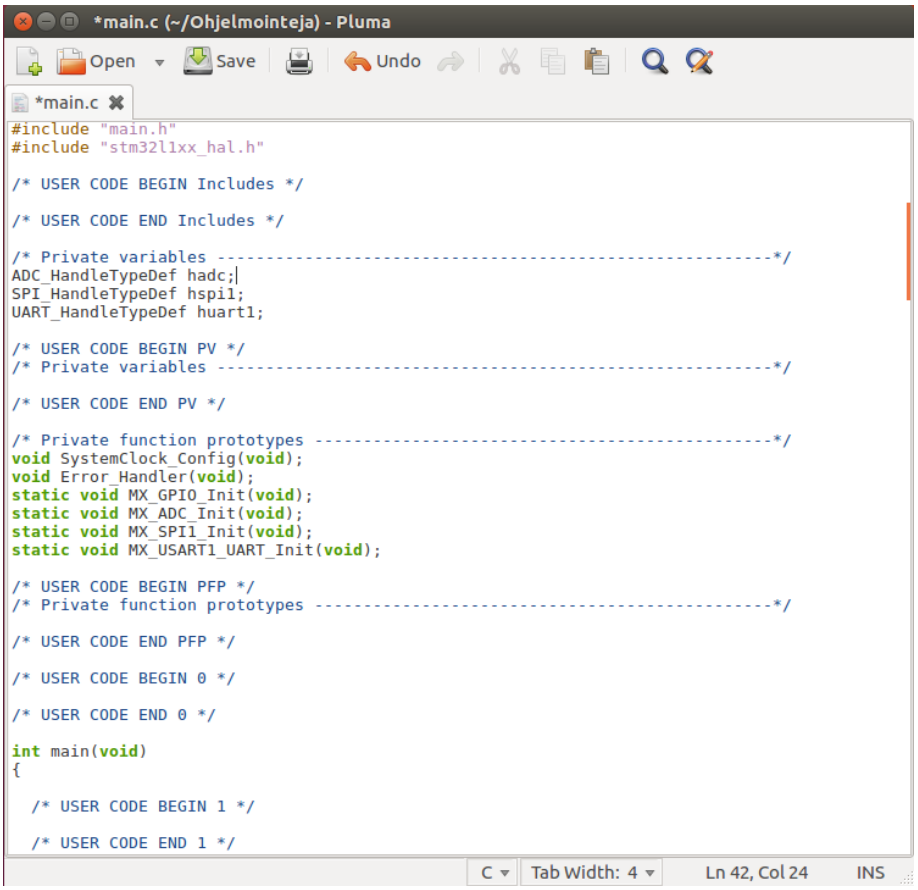


KUVA 10. STM32CubeMX-projektin asetussivu.

Kun kaikki tarvittavat asetukset oli tehty, voitiin generoida alustuskoodi (liite 3).

4.3 Ohjelmakoodin kirjoitus

Ohjelmakoodi kirjoitettiin Pluma-tekstieditorilla (kuva 11). Ohjelmakoodin pohjana toimi STM32CubeMX:llä tehty alustuskoodi, jossa oli määritelty kommentein paikat, johon käyttäjä kirjoittaa oman koodinsa. Oma koodi oli hyvä kirjoittaa näihin väleihin, koska STM32CubeMX-ohjelma voidaan asettaa säilyttämään koodi näissä kohdissa, mikäli työkalulla halutaan vielä muuttaa alustuskoodia.



KUVA 11. Näkymä Pluma-tekstieditorista.

CC1200-ajuriohjelmiston kirjoituksessa otettiin mallia Texas Instrumentsin tarjoamasta esimerkkiohjelmasta MSP430-mikrokontrollerille. Esimerkkiohjelmassa käytettyjen mikrokontrollerin oheislaitteiden käytössä oli eroja työssä käytettyihin, joten varsinainen ajuriohjelmisto täytyi kirjoittaa kokonaan uusiksi.

Työssä käytetylle kiihtyvyysanturille löytyi internetistä toisen ohjelmoijan tekemä vapaaseen käyttöön tarkoitettu esimerkkiohjelma, jonka pohjalta lähdettiin ajuriohjelmistoa tekemään. Tässäkin esimerkissä käytetyn mikrokontrollerin ohjaus poikkesi suuresti työssäkäytettyyn, joten ajuriohjelmisto täytyi kirjoittaa itse uudelleen.

Lämpötila-kosteusanturin ajuriohjelmistoon ei tarvinnut työssä käyttää paljoa aikaa, koska yrityksellä oli jo samasta anturista ajuriohjelma, joka toimi työssäkäytetyn mikrokontrollerin kanssa pienin muutoksin.

4.4 GNU make:n ja GCC:n käyttö

Työssä käytettiin GNU make -ohjelmaa, joka määrittelee automaattisesti, mitkä osat ohjelmasta täytyy kääntää. GNU makea varten täytyi tehdä makefile, johon määriteltiin mm. käännettävät tiedostot, mitä toolchainia käytetään, käytettävän linker skriptin ja kääntäjän asetukset. Esimerkki makefilestä liitteessä 3.

Kun halutunlainen makefile saatiin tehtyä, asetettiin se projektikansioon. GNU makea käytettiin navigoimalla linuxin komentorivillä projektikansioon ja antamalla komento make. Make komennolla GNU make suoritti kansiossa olevan makefilen, jonka perusteella ohjelma kääntyi ja ohjelma saatiin binääritiedostoon. Mikäli projektiin lisättiin C-tiedostoja ohjelman eri versioiden kääntämisen välillä, täytyi ne lisätä makefileen.

4.5 Laitteen ohjelmointi

Laitetta voitiin lähteä ohjelmoimaan, kun GCC:llä oli saatu onnistuneesti käännettyä ohjelmakoodi binäärimuotoon. Binääritiedoston ohjelmoiminen mikrokontrollerin ohjelmamuistiin tapahtui stm32flash-ohjelmalla. Aina kun mikrokontrollerille haluttiin ajaa uusi ohjelma, täytyi mikrokontrolleri asettaa ohjelmointitilaan. Mikrokontrollerin asettaminen ohjelmointitilaan tapahtui painamalla resetointi nappia samalla kun korttiin kytkettiin virta.

5 TESTAUS

Työssä korttien välistä radioyhteyttä testattiin kantomatkan ja pakettien eheyden osalta, joihin kirjoitettiin omat testausohjelmat. Kiihtyvyysanturille ja lämpötila-kosteusanturille tehtiin myös omat testausohjelmansa. Testauksista työssä ei ollut tarpeen tehdä pöytäkirjaa eikä standardisoituja olosuhteita.

5.1 Kantomatkan testaus

Kantomatkan testausta varten kirjoitettiin ohjelma, jossa kortin ledi välähti aina, kun paketti oli vastaanotettu. Toinen kortti asetettiin lähettämään paketteja tasaisin väliajoin. Testaus tapahtui jättämällä paketteja lähettävä kortti paikalleen ja kulkemalla toisen kortin kanssa eri paikkoihin. Kun kortin ledi lopetti vilkkumisen tai vilkkumistahti hidastui, voitiin olettaa, että kantama alkaa olla liian pitkä tai kohdassa on signaalin katvealue.

5.2 Datapakettien eheyden testaus

Datapakettien eheyttä testattiin kirjoittamalla toiselle kortille ohjelma, joka tulostaa tietokoneen sarjaporttiin vastaanotetun paketin. Toiselle korteista kirjoitettiin ohjelma, joka lähettää ennalta määrättyjä paketteja ja näin voitiin vertailla, onko vastaanotettu data virheetöntä.

5.3 Kiihtyvyysanturin ja lämpötila-kosteusanturin testaus

Kiihtyvyysanturin ja lämpötila-kosteusanturin testaukseen kirjoitettiin ohjelmat, jotka tulostivat tietokoneen sarjaporttiin anturien antamat arvot aina kun kortilla ollutta nappia painettiin.

Kiihtyvyysanturia testattaessa korttia pyöriteltiin eri asentoihin ja painettiin nappia, jolloin kiihtyvyysdata tulostui sarjaporttiin. Kiihtyvyysanturilta tulostetut arvot korreloivat hyvin kortin asentoa suhteessa maanvetovoimaan. Datalle ei ollut työssä tarvetta tehdä tarkempaa verifiointia.

Lämpötila-kosteusanturin testaus suoritettiin painamalla nappia ja vertaamalla tulostettuja arvoja sen hetkiseen ilman lämpötilaan ja kosteuteen. Anturin antamat tulokset olivat lähellä vallitsevia olosuhteita. Anturia testattiin myös hönkimällä siihen hengitysilmaa ja arvot alkoivat vastaavasti nousta, joten voitiin todeta, että lämpötila-kosteusanturin ajuriohjelmisto toimi.

6 YHTEENVETO

Työn tavoitteena oli toteuttaa Texas Instrumentsin CC1200-radiopiirin ajurit STMicroelectronicsin STM32L151RC-mikrokontrollerille. Työssä tehtiin myös ajurit kiihtyvyysanturille ja lämpötila-kosteusanturille.

Työ aloitettiin tutustumalla Linux-ympäristöön ja GCC-kääntäjään. Kun mikrokontrollerin ohjelmointi luonnistui ja perustoiminnot saatiin haltuun, voitiin alkaa perehtyä radiopiirin toimintaan ja sen ohjelmointiin. Työn haastavimmaksi osaksi osoittautui GCC-kääntäjän käyttäminen, koska opintojen aikana kaikki käytetyt työkalut olivat hoitaneet ohjelmakoodien kääntämisen eikä asiaa ollut sen erityisemmin käsitelty. Työssä kului enemmän aikaa datalehtien sekä muiden dokumenttien lukemiseen ja työkalujen ja ympäristön opiskeluun kuin itse ajuriohjelmistojen kirjoitukseen.

Työssä tehdyt ajuriohjelmistot antavat yksinkertaisen ja joustavan pohjan sovelluskohtaiselle radiopiirin käytölle sovellusohjelmistossa. Radiopiirin ajuriohjelmistolla voitiin asettaa ja lukea radiopiirin rekisterit. Tämän lisäksi pienellä sovellusohjelmalla kahden kortin välinen kommunikaatio toimi ja näin ollen työlle asetetut vaatimukset saatiin täytettyä. Tässä työssä ei tehty laajempaa sovellusohjelmistoa kortille, koska se ei ollut työn tavoitteenakaan.

LÄHTEET

ARM. (2017). *CMSIS*. Haettu 8. 4 2017 osoitteesta CMSIS: https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

Doulos. (2017). *CMSIS*. Haettu 8. 4 2017 osoitteesta CMSIS: https://www.doulos.com/knowhow/arm/CMSIS/CMSIS_Doulos_Tutorial.pdf

Gast, M. S. (2005). Teoksessa M. S. Gast, *802.11 Wireless Networks: The Definitive Guide, Second Edition.*

GNU make. (2016). Noudettu osoitteesta https://www.gnu.org/software/make/manual/make.html#Overview

Hirvonen, J. (ei pvm). *RF-mikrokontrollerikortti.*

nitsky. (2013). Haettu 20. 4 2017 osoitteesta makefile example: https://github.com/nitsky/stm32-example/blob/master/Makefile

stm32flash. (2016). *stm32flash*. Haettu 8. 4 2017 osoitteesta stm32flash: https://sourceforge.net/p/stm32flash/wiki/Home/

STMicroelectronics. (2016a). *STM32CubeMX*. Haettu 6. 4 2017 osoitteesta STM32CubeMX: http://www.st.com/en/development-tools/stm32cubemx.html

STMicroelectronics. (2016b). *STM32L1 Series*. Haettu 7. 4 2017 osoitteesta http://www.st.com/en/microcontrollers/stm32l1-series.html?querycriteria=productId=SS1295

STMicroelectronics. (2016c). *STM32L1 HAL*. Haettu 6. 4 2017 osoitteesta STM32L1 HAL: http://www.st.com/content/ccc/resource/technical/document/user_manual/97/4d/5f/9a/ed/e4/4e/66/DM00132099.pdf/files/DM00132099.pdf/jcr:content/translations/en.DM00132099.pdf

STMicroelectronics. (2016d). *STM32L151RC*. Haettu 7. 4 2017 osoitteesta STM32L151RC: http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32l1-series/stm32l151-152/stm32l151rc.html

STMicroelectronics. (2017). *STM32CubeMX*. Haettu 6. 4 2017 osoitteesta STM32CubeMX: http://www.st.com/content/ccc/resource/technical/document/user_manual/10/c5/1a/43/3a/70/43/7d/DM00104712.pdf/files/DM00104712.pdf/jcr:content/translations/en.DM00104712.pdf

Techiessms. (2016). Haettu 5. 4 2017 osoitteesta http://techiesms.blogspot.fi/2016/04/practical-application-modulation.html

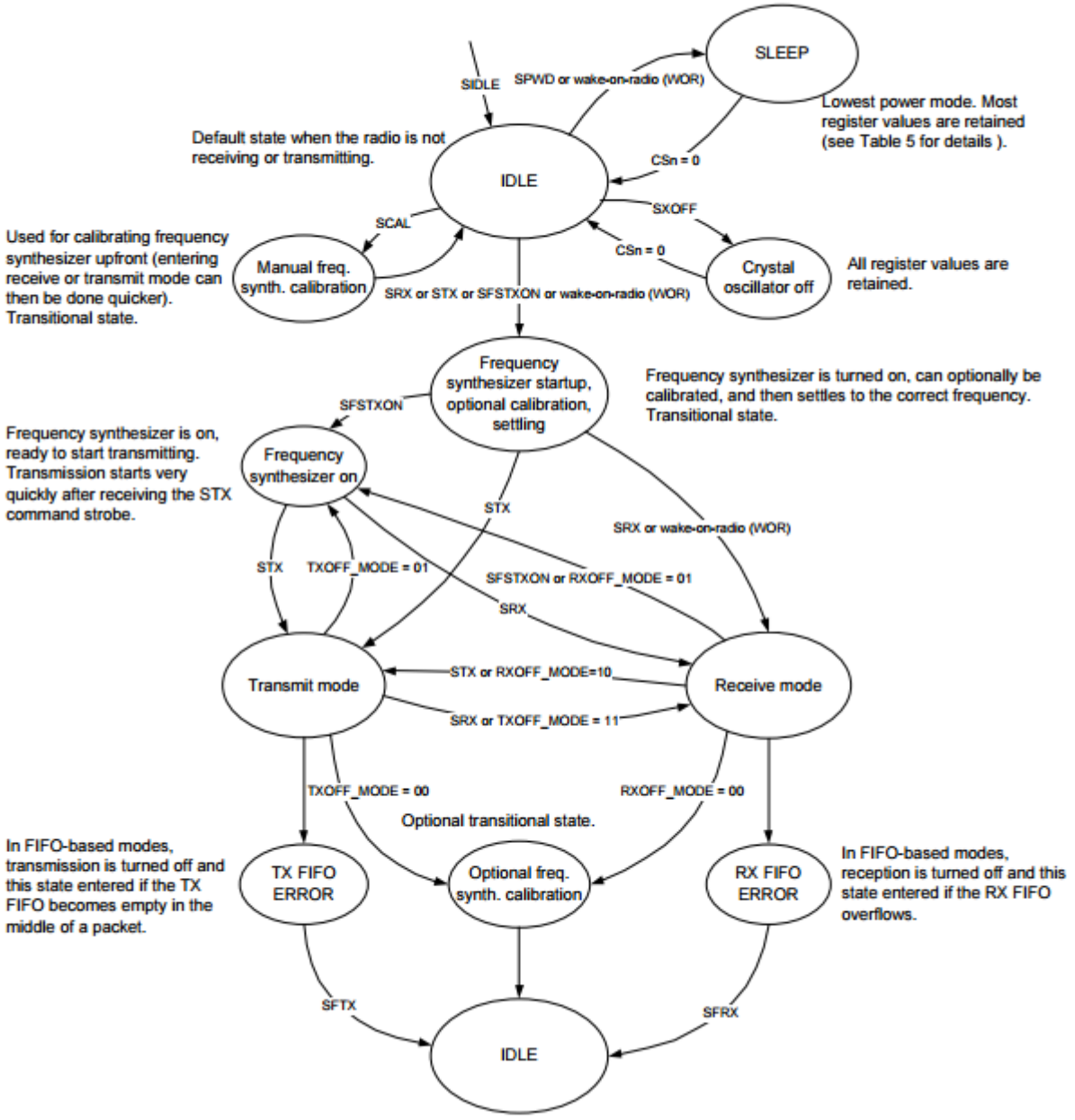
Texas Instruments. (2013a). Haettu 4. 4 2017 osoitteesta CC1200 Users guide: http://www.ti.com/lit/ug/swru346b/swru346b.pdf

Texas Instruments. (2013b). Haettu 4. 4 2017 osoitteesta http://www.ti.com/product/CC1200/description

Viestintävirasto. (2016). *Taajuustaulukko*. Haettu 8. 4 2017 osoitteesta https://www.viestintavirasto.fi/attachments/maaraykset/Taajuusjakotaulukko_22.12.2016.pdf

LIITE 1: RADION TOIMINTAKAAVIO

CC120X



LIITE 2: RADION GPIO SIGNAALIT

GPIOx_CFG	Signal Name	Description
0	RXFIFO_THR	Associated to the RX FIFO. Asserted when the RX FIFO is filled above FIFO_CFG.FIFO_THR. De-asserted when the RX FIFO is drained below (or is equal) to the same threshold. This signal is also available in the MODEM_STATUS1 register
1	RXFIFO_THR_PKT	Associated to the RX FIFO. Asserted when the RX FIFO is filled above FIFO_CFG.FIFO_THR or the end of packet is reached. De-asserted when the RX FIFO is empty
2	TXFIFO_THR	Associated to the TX FIFO. Asserted when the TX FIFO is filled above (or is equal to) (127 –FIFO_CFG.FIFO_THR). De-asserted when the TX FIFO is drained below the same threshold. This signal is also available in the MODEM_STATUS0 register
3	TXFIFO_THR_PKT	Associated to the TX FIFO. Asserted when the TX FIFO is full. De-asserted when the TX FIFO is drained below (127 – FIFO_CFG.FIFO_THR)
4	RXFIFO_OVERFLOW	Asserted when the RX FIFO has overflowed. De-asserted when the RX FIFO is flushed (see Section 3.2.4). This signal is also available in the MODEM_STATUS1 register
5	TXFIFO_UNDERFLOW	Asserted when the TX FIFO has underflowed. De-asserted when the TX FIFO is flushed (see Section 3.2.4). This signal is also available in the MODEM_STATUS0 register
6	PKT_SYNC_RXTX	RX: Asserted when sync word has been received and de-asserted at the end of the packet. Will de-assert when the optional address and/or length check fails or the RX FIFO overflows/underflows. TX: Asserted when sync word has been sent, and de-asserted at the end of the packet. Will de-assert if the TX FIFO underflows/overflows
7	CRC_OK	Asserted simultaneously as PKT_CRC_OK. De-asserted when the first byte is read from the RX FIFO
8	SERIAL_CLK	Serial clock (RX and TX mode). Synchronous to the data in synchronous serial mode. Data is set up on the falling edge in RX and is captured on the rising edge of the serial clock in TX
9	SERIAL_RX	Serial data (RX mode). Used for both synchronous and transparent mode. Synchronous serial mode: Data is set up on the falling edge. Transparent mode: No timing recovery (outputs just the hard limited baseband signal)
10		Reserved (used for test)
11	PQT_REACHED	Preamble Quality Reached. Asserted when the quality of the preamble is above the programmed PQT value (see Section 6.8). This signal is also available in the MODEM_STATUS1 register
12	PQT_VALID	Preamble quality valid. Asserted when the PQT logic has received a sufficient number of symbols (see Section 6.8). This signal is also available in the MODEM_STATUS1 register
13	RSSI_VALID	RSSI calculation is valid
14		RSSI Signals
	3 RSSI_UPDATE	A pulse occurring each time the RSSI value is updated (see Figure 16)
	2 RSSI_UPDATE	A pulse occurring each time the RSSI value is updated (see Figure 16)
	1 AGC_HOLD	AGC waits for gain settling (see Figure 16)
	0 AGC_UPDATE	A pulse occurring each time the front end gain has been adjusted (see Figure 16)
15		Clear channel assessment
	3 CCA_STATUS	Current CCA status
	2 TXONCCA_DONE	A pulse occurring when a decision has been made as to whether the channel is busy or not. This signal must be used as an interrupt to the MCU. When this signal is asserted/de-asserted, TXONCCA_FAILED can be checked
	1 CCA_STATUS	Current CCA status
	0 TXONCCA_FAILED	TX on CCA failed. This signal is also available in the MARC_STATUS0 register
16	CARRIER_SENSE_VALID	CARRIER_SENSE is valid (see Figure 16)
17	CARRIER_SENSE	Carrier sense. High if RSSI level is above threshold (see Section 6.9.1) (see Figure 16)

18			DSSS signals for DSSS repeat mode (RX). MODCFG_DEV_E.MODEM_MODE = 1
	3	DSSS_CLK	DSSS clock (see Section 5.2.6 for more details)
	2	DSSS_DATA0	DSSS data0 (see Section 5.2.6 for more details)
	1	DSSS_CLK	DSSS clock (see Section 5.2.6 for more details)
	0	DSSS_DATA1	DSSS data1 (see Section 5.2.6 for more details)
19	PKT_CRC_OK		Asserted in RX when PKT_CFG1.CRC_CFG = 1 or 10_b and a good packet is received. This signal is always on if the radio is in TX or if the radio is in RX and PKT_CFG1.CRC_CFG = 0. The signal is de-asserted when RX mode is entered and PKT_CFG1.CRC_CFG ≠ 0. This signal is also available in the LQI_VAL register
20	MCU_WAKEUP		MCU wake up signal. Read MARC_STATUS1.MARC_STATUS_OUT to find the cause of the wake up event (see Section 3.4.1.2 for more details). This signal is also available in the MARC_STATUS0 register. The signal is a pulse
21	SYNC_LOW0_HIGH1		DualSync detect. Only valid when SYNC_CFG1.SYNC_MODE = 111_b. When SYNC_EVENT is asserted this bit can be checked to see which sync word is found. This signal is also available in the DEM_STATUS register
22	3		Reserved (used for test)
	2		Reserved (used for test)
	1		Reserved (used for test)
	0	AES_COMMAND_ACTIVE	Indicates that an AES command is being executed
23	LNA_PA_REG_PD		Common regulator control for PA and LNA. Indicates RF operation
24	LNA_PD		Control external LNA[4]
25	PA_PD		Control external PA[4]
26	RX0TX1_CFG		Indicates whether RF operation is in RX or TX (this signal is 0 in IDLE state)
27			Reserved (used for test)
28	IMAGE_FOUND		Image detected by image rejection calibration algorithm
29	CLKEN_CFM		Data clock for demodulator soft data (see Section 5.2.4 for more details)
30	CFM_TX_DATA_CLK		Data clock for modulator soft data (see Section 5.2.4 for more details)
31 - 32			Reserved (used for test)
33	RSSI_STEP_FOUND		RSSI step found during packet reception (after the assertion of SYNC_EVENT). The RSSI step is either 10 or 16 dB (configured through AGC_CFG1.RSSI_STEP_THR). This signal is also available in the DEM_STATUS register
34	3	AES_RUN	AES enable. This signal is asserted as long as the AES module is enabled given that AES.AES_ABORT = 0. This signal is also available in the AES register
	2	AES_RUN	Same as 3
	1	RSSI_STEP_EVENT	RSSI step detected. This signal is asserted if there is an RSSI step of 3 or 6 dB during sync search or if there is an RSSI step of 10 or 16 dB during packet reception. The RSSI step is configured through AGC_CFG1.RSSI_STEP_THR). The signal is a pulse
	0	RSSI_STEP_EVENT	Same as 1
35	3		Reserved (used for test)
	2		Reserved (used for test)
	1	LOCK	Out of lock status signal. Indicates out of lock when the signal goes low. This signal is also available in the FSCAL_CTR register
	0	LOCK	Same a 1
36	ANTENNA_SELECT		Antenna diversity control. Can be used to control external antenna switch. If differential signal is needed, two GPIOs can be used with one of them having IOCFGx.GPIOx_INV set to 1
37	MARC_2PIN_STATUS[1]		Partial MARC state status. These signals are also available in the MARCSTATE register

MARC_2PIN_STATUS[1]	MARC_2PIN_STATUS[0]	State
0	0	SETTLING
0	1	TX
1	0	IDLE
1	1	RX

38	MARC_2PIN_STATUS[0]		See MARC_2PIN_STATUS[1]
39	3		Reserved (used for test)
	2	TXFIFO_OVERFLOW	Asserted when the TX FIFO has overflowed. De-asserted when the TX FIFO is flushed (see Section 3.2.4). This signal is also available in the MODEM_STATUS0 register
	1		Reserved (used for test)
	0	RXFIFO_UNDERFLOW	Asserted when the RX FIFO has underflowed. De-asserted when the RX FIFO is flushed (see Section 3.2.4). This signal is also available in the MODEM_STATUS1 register
40	3	MAGN_VALID	New CORDIC magnitude sample
	2	CHFILT_VALID	New channel filter sample
	1	RCC_CAL_VALID	RCOSC calibration has been performed at least once
	0	CHFILT_STARTUP_VALID	Channel filter has settled
41	3	COLLISION_FOUND	Asserted if a new preamble is found and the RSSI has increased 10 or 16 dB during packet reception. (MDMCFG1.COLLISION_DETECT_EN = 1) This signal is also available in the DEM_STATUS register
	2	SYNC_EVENT	Sync word found (pulse)
	1	COLLISION_FOUND	Same as 3
	0	COLLISION_EVENT	Preamble found during receive with an RSSI step of 10 or 16 dB (pulse)
42	PA_RAMP_UP		Asserted when ramping is started (for compliance testing)
43	3	CRC_FAILED	Packet CRC error
	2	LENGTH_FAILED	Packet length error
	1	ADDR_FAILED	Packet address error
	0	UART_FRAMING_ERROR	Packet UART framing error
44	AGC_STABLE_GAIN		AGC has settled to a gain. The AGC gain is reported stable whenever the current gain setting is equal to the previous gain setting. This condition is evaluated each time a new internal RSSI estimate is computed (see Figure 16)
45	AGC_UPDATE		A pulse occurring each time the front end gain has been adjusted (see Figure 16)
46			ADC data (test purposes only)
	3	ADC_CLOCK	ADC clock
	2	ADC_Q_DATA_SAMPLE	ADC sample (Q data)
	1	ADC_CLOCK	ADC clock
	0	ADC_I_DATA_SAMPLE	ADC sample (I data)
47			Reserved (used for test)
48	HIGHZ		High impedance (tri-state)
49	EXT_CLOCK		External clock (divided crystal clock). The division factor is controlled through the ECG_CFG.EXT_CLOCK_FREQ register field
50	CHIP_RDYn		Chip ready (XOSC is stable)
51	HW0		HW to 0 (HW to 1 achieved with IOCFGx.GPIOx_INV = 1)
52 - 53			(Reserved (used for test)
54	CLOCK_40K		40 kHz clock output from internal RC oscillator
55	WOR_EVENT0		WOR EVENT0
56	WOR_EVENT1		WOR EVENT1
57	WOR_EVENT2		WOR EVENT2
58			Reserved (used for test)
59	XOSC_STABLE		XOSC is stable (has finished settling)
60	EXT_OSC_EN		External oscillator enable (used to control e.g. a TCXO). Note that this signal is only asserted is a TCXO is present
61 - 63			Reserved (used for test)

LIITE 3: MAKEFILE ESIMERKKI

```
BUILDDIR=build


DEVICE = stm32/device
CORE = stm32/core
PERIPH = stm32/periph
DISCOVERY = stm32/discovery
USB = stm32/usb


SOURCES += $(DISCOVERY)/src/stm32f3_discovery.c


SOURCES += $(PERIPH)/src/stm32f30x_gpio.c \
           $(PERIPH)/src/stm32f30x_i2c.c \
           $(PERIPH)/src/stm32f30x_rcc.c \
           $(PERIPH)/src/stm32f30x_spi.c \
           $(PERIPH)/src/stm32f30x_exti.c \
           $(PERIPH)/src/stm32f30x_syscfg.c \
           $(PERIPH)/src/stm32f30x_misc.c


SOURCES += startup_stm32f30x.s
SOURCES += stm32f30x_it.c
SOURCES += system_stm32f30x.c
SOURCES += main.c
OBJECTS = $(addprefix $(BUILDDIR)/, $(addsuffix .o, $(basename $(SOURCES))))
INCLUDES += -I$(DEVICE)/include \
            -I$(CORE)/include \
            -I$(PERIPH)/include \
            -I$(DISCOVERY)/include \
            -I$(USB)/include \
            -I\
ELF = $(BUILDDIR)/program.elf
HEX = $(BUILDDIR)/program.hex
BIN = $(BUILDDIR)/program.bin
CC = arm-none-eabi-gcc
LD = arm-none-eabi-gcc
AR = arm-none-eabi-ar
OBJCOPY = arm-none-eabi-objcopy

CFLAGS  = -O0 -g -Wall -I.\
   -mcpu=cortex-m4 -mthumb \
   -mfpu=fpv4-sp-d16 -mfloat-abi=hard \
   $(INCLUDES) -DUSE_STDPERIPH_DRIVER
```

```
LDSCRIPT = stm32_flash.ld
LDFLAGS += -T$(LDSCRIPT) -mthumb -mcpu=cortex-m4 -nostdlib
$(BIN): $(ELF)
	$(OBJCOPY) -O binary $< $@
$(HEX): $(ELF)
	$(OBJCOPY) -O ihex $< $@
$(ELF): $(OBJECTS)
	$(LD) $(LDFLAGS) -o $@ $(OBJECTS) $(LDLIBS)
$(BUILDDIR)/%.o: %.c
	mkdir -p $(dir $@)
	$(CC) -c $(CFLAGS) $< -o $@


$(BUILDDIR)/%.o: %.s
	mkdir -p $(dir $@)
	$(CC) -c $(CFLAGS) $< -o $@
flash: $(BIN)
	st-flash write $(BIN) 0x8000000
clean:
	rm -rf build
```

Liite 3. (nitsky, 2013)

LIITE 4: STM32CUBEMX:N GENEROIMA ALUSTUSKOODI

```
/**
  ******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  ******************************************************************************
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
```

```c
#include "main.h"
#include "stm32l1xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();

  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
```

```
  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC init function */
static void MX_ADC_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
conversion)
    */
  hadc.Instance = ADC1;
```

```
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  hadc.Init.LowPowerAutoWait = ADC_AUTOWAIT_DISABLE;
  hadc.Init.LowPowerAutoPowerOff = ADC_AUTOPOWEROFF_DISABLE;
  hadc.Init.ChannelsBank = ADC_CHANNELS_BANK_A;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.NbrOfConversion = 1;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and
its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_4CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* SPI1 init function */
static void MX_SPI1_Init(void)
{

  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_HalfDuplex_Init(&huart1) != HAL_OK)
```

```
  {
    Error_Handler();
  }

}

/** Pinout Configuration
*/
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```