TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

NB-IOT RAADIOSIDE KASUTAMINE KESKKONNA PARAMEETRITE MÕÕTMISEKS

USING NB-IOT RADIO COMMUNICATION TO MEASURE ENVIRONMENTAL PARAMETERS

TELEMAATIKA JA ARUKATE SÜSTEEMIDE ÕPPEKAVA LÕPUTÖÖ

Üliõpilane: Roald Põllu
Üliõpilaskood: 193238EDTR
Juhendaja: Sergei Pavlov, lektor

Kohtla-Järve, 2022

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"...." ..... 20.....

Autor: .....
/ allkiri /

Töö vastab rakenduskõrgharidusõppe lõputööle/magistritööle esitatud nõuetele
"...." ..... 20.....

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud
"...." ..... 20.....

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS

Mina Roald Põllu (sünnikuupäev:12.12.1986)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

NB-IoT raadioside kasutamine keskkonna parameetrite mõõtmiseks, mille juhendaja on Sergei Pavlov,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

TalTech Inseneriteaduskond Virumaa kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Roald Põllu, 193238EDTR

Õppekava, peaeriala: EDTR17/18 - Telemaatika ja arukad süsteemid

Juhendaja(d): Lektor, Sergei Pavlov, sergei.pavlov@taltech.ee

Lõputöö teema:

(eesti keeles) NB-IoT raadioside kasutamine keskkonna parameetrite mõõtmiseks

(inglise keeles) Using NB-IoT radio communications to measure environmental parameters

Lõputöö põhieesmärgid:

Lõputöö põhieesmärgiks on uurida kuidas saab edastada keskkonnaparameetreid raadiosidet kasutades arendusplaadiga nRF9160DK.

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Tutvuda arendusplaadi nRF9160DK dokumentatsiooniga	01.12
2.	Leida viis kuidas edastada keskkonnaparameetreid	01.12
3.	Lisada arendusplaadile andur ens210 keskkonna parameetrite mõõtmiseks	01.12
4.	Lisatud anduri parameetrite edastamine üle raadioside	01.12
5.	Andmete visualiseerimine	01.12
6.	ToF, ultraheli-ja vingugaasianduri integreerimine	01.12

Töö keel: eesti keel **Lõputöö esitamise tähtaeg:** "....."..... 20.....a

Üliõpilane: Roald Põllu "....."..... 20.....a
/allkiri/

Juhendaja: Sergei Pavlov "....."..... 20.....a
/allkiri/

Programmijuht: Žanna Gratšjova "....."..... 20.....a
/allkiri/

SISUKORD

EESSÕNA 7
LÜHENDITE JA TÄHISTE LOETELU 8
SISSEJUHATUS 10
1 RAADIOSIDEVÕRK 11
1.1 NB-IoT 11
2 PROTOKOLLID 12
2.1 MQTT 12
2.2 I2C 13
3 ARENDUSPLAAT 14
3.1 Riistvara 14
3.2 Tarkvarad ja programmid 16
3.2.1 Programmer 17
3.2.2 LTE Link Monitor 17
3.2.3 NRF Connect SDK ja Visual Studio Code 18
3.3 Arendusplaadi nRF91 seeria võgurežiim 19
4 ANDURID 20
4.1 Temperatuuri- ja niiskuseandur ENS210 20
4.2 Kauguse ja liigutuste tuvastamise andur VL53L0X (ToF) 20
4.3 Ultraheli moodul HC-SR04 21
4.4 Vingugaasiandur MQ-7 21
5 AWS (AMAZON WEB SERVICES) 23
6 TEHTUD KATSED JA TULEMUSED ANDURIGA ENS210 25
6.1 Arendusplaadi ja anduri sidumine 26
6.2 Keskkonnaandmete edastamine AWS IoT Core 27
6.3 Andmete salvestamine Amazon Timestream andmebaasis 31
6.4 Andmete visualiseerimine 33
7 ANDURITE HC-SR04, VL53L0X JA MQ-7 INTEGREERIMINE 35
7.1 Ultraheliandurи HC-SR04 integreerimine digitaalse sisendi ja väljundiga 38
7.2 ToF anduri VL53L0X integreerimine I2C liiniga 39
7.3 Vingugaasianduri MQ-7 integreerimine analoogsisendiga 39
8 TULEMUSED 41
9 PROJEKTI EDASIARENDAMISE VÕIMALUSED 42
KOKKUVÕTE 43

SUMMARY 44
KASUTATUD KIRJANDUSE LOETELU 46
Lisa 1 - AWS_iot mall SDK vrsioonist 2.1.0 49
Lisa 2 - ENS210 kood kasutades SDK 2.1.0 draivereid 61
Lisa 3 - HC-SR04 kood [34] 62
Lisa 4 - ToF anduri VL53L0X kood kasutades SDK 2.1.0 draivereid 65
Lisa 5 - Vingugaasianduri kood [32] [33] 66
Lisa 6 - AWS kood [32] [33] [34] 68
Lisa 7 - Näidikute paneelid Grafanas 80

EESSÕNA

Diplomitöös autor kasutas Nordic Semiconductori arendusplaati *nRF9160 DK* mis on mõeldud just *IoT* seadmete arendamiseks. Arendusplaadiga kasutatav SDK võimaldab luua, katsetada ja arendada erinevaid programme, millede hulka kuulub ka keskkonnaparameetrite edastamine. Programmeerimise keeleks on C ja VisualStudio Code, millele on installeeritud vastav SDK. Peale mille on kasutatud ka erinevate tootjate andureid mis on juhtmetega ühendatud arendusplaadi erinevate sisendite ja väljunditega.

Diplomitöö teema ja etapid selle läbimiseks kujunesid töö käigus juhendajalt saadud infot ja soovitusi järgides. Töö käigus uuritakse seadme dokumentatsiooni, kuidas näeb välja riistvaraline pool. Kuidas peab seadme ette valmistama, et saaks alustada programmeerimisega ja kuidas programmeeritakse arendusplaati ja milliseid programme on vaja kasutada töö tegemiseks, samuti ka vaja teadmisi elektrotehnikast.

Andmete edastamiseks kasutatakse *MQTT* protokolli mis on tänapäeval *IoT* seadmete suhtluses üsna populaarne ja väikese koodijalajäljega. Andmete saamiseks kasutatakse erinevaid andureid mis kasutavad digitaalseid sisendeid ja väljundeid, analoog sisendeid ja ka *I2C* protokolli suhtlemaks arendusplaadiga. Veel kasutatakse AWS-i erinevaid keskkondi kus on võimalik andmed salvestada ja esitada graafikutena.

Tänu soovin avaldada enda juhendajale Sergei Pavlovile kes mind selle töö juures aitas nii seadmetega kui teema valikuga ning esitas mulle väljakutse lahendada antud ülesanne.

Märksõnadeks on IoT, MQTT, C programmeerimine, NB-IoT, diplomitöö.

LÜHENDITE JA TÄHISTE LOETELU

NB-IoT – kitsaribaline asjade Internet (NB-IoT), mis on mõeldud IoT seadmetele 200 kHz ribalaiusega, mis on määratletud 3GPP versioonis 13 [1]

I/O (input/output) – sisend ja väljund

MQTT – IoT sõnumside standard [2]

IoT (Internet Of Things) – asjade internet

AWS (Amazon Web Services) – pilveplatvorm

LTE (Long Term Evolution) – See on traadita andmeedastuse standard, mis võimaldab andmeid väga kiiresti alla laadida [3]

LTE-M – LTE võrgu standard mi son mõeldus IoT seadmetele [1]

I2C – jadaliini sideprotokoll

UART (Universal Asynchronous Receiver/Transmitter) – universaalne asünkroonne vastuvõtja/saatja [1]

MCU (Microcontroller Unit) – väike arvuti mikroskeemis [1]

kHz – kilohertz, sageduse mõõtühik

UE (User Equipment) – kasutajaseadmed [1]

B1- B5, B8, B12 - B14, B17 - B20, B25, B26, B28 ja B66 – LTE võrgud [1]

GPIO (General-Purpose Input/Output) – digitaalse signaali väljaviik, mida saab kasutada sisend/väljundina või mõlemana [1]

SPI – liides välise mälu jaoks arendusplaadil nRF9160 DK [4]

MB – megabait, digitaalsete andmete mõõtühikUART

kB – kilobait, digitaalsete andmete mõõtühik

RAM – operatiivmälu

I2S (Inter-IC sound) – elektrilise jadaliidese standard, mida kasutatakse digitaalsete heliseadmete ühendamiseks [4]

V – elektripinge mõõtühik

SoC (System on Chip) – Mikrokiip, mis integreerib kõik vajalikud arvuti või muude elektroonikasüsteemide elektroonilised vooluringid ja komponendid ühte integraallülitusse [1]

API (Application Programming Interface) – Keele ja sõnumi formaat, mida rakendusprogramm kasutab operatsioonisüsteemi, rakenduse või muude teenustega suhtlemiseks [1]

GSM (*Global System* for *Mobile* Communications) – Globaalne mobiilsidesüsteem [5]

TCP/IP – andmeedastuse võrgumudel [6]

WSS – kahesuunaline täisdupleksprotokoll mida kasutatakse kliendi ja serveri suhtluses [7]

HTTPS – turvalisele veebiserverile juurdepääsuks kasutatav protokoll [8]

RTOS (real-time operating system)– reaalaja operatsioonisüsteem [9]

JSON (JavaScript Object Notation) – programmeerimis keelest sõltumatu andmevorming, mis on inimesele kergesti loetav [10]

USB (universal serial bus) – universaalne jadasiin mis võimaldab arvutil suhelda väliste seadmetega ja ka toita neid [11]

SQL (Structured Query Language) – Struktureeritud päringukeel on standardiseeritud programmeerimiskeel, mida kasutatakse relatsiooniandmebaaside haldamiseks ja nendes sisalduvate andmetega erinevate toimingute tegemiseks [12]

ToF – Time-of-Flight [13]

SDA (Serial Data) – Ülem- ja alamseadme liin andmete saatmiseks ja vastuvõtmiseks [14]

SCL (Serial Clock) – Taktsageduse liin [14]

ACK – Signaali nimi, et andmed on edukalt vastu võetud [15]

NACK – Signaali nimi, et andmed ei ole edukalt vastu võetud [15]

ADC (analog-to-digital converter)– Analoog digitaal konverter

CoAP (Constrained Application Protocol) - Piiratud rakenduse protokoll [16]

LoRaWAN (Long Range Wide Area Network)- Laiaulatuslik laivõrk [17]

SISSEJUHATUS

Antud lõputöö on aktuaalne kuna *IoT* seadmete kasutamine tänapäeva maailmas on peaaegu igas elu valdkonnas ja *IoT* seadmete osakaal aina kasvab ja vajab ka energiasäästlike lahendusi kohtades kus inimestel on raskem ise püsivalt kohapeal olla või ei ole lihtsalt otstarbekas ja seega on oluline kasutada lahendusi mis võimaldavad väikeste voolutarvetega seda teha.

Lõputöös kasutatakse *NB-IoT* mobiilside võrku mis on mõeldud spetsiaalselt *IoT* seadmetele. Antud võrk aitab luua turvalise ja kiire suhtluse seadme ja lõppkasutaja vahel, mis omakorda võimaldab luua erinevaid targa linna lahendusi. Võrk ise on litsentseeritud ja see seab võrgule teatud turvalisuse tingimused.

NB-IoT võrgu kattuvus on üle Eesti, samuti leviala on parem, kuna on võimalik kasutada seadmeid nii maa all, vee all ja teistes sarnastes kohtades kus tavamobiilivõrgul levi on kehv või puudub. Näiteks võib tuua prügikonteinerite täituvuse mõõtmine, mis aitaks kindlasti paremini optimeerida ressursse ja hoida kütusekulu kokku prügiautodel. Samamoodi võib kasutada sadeveetorude veetaseme mõõtmiseks linnades, et vältida või ennetada uputusi paremini. Kasutusvaldkondi kus antud tehnoloogiat saab kasutada ja hoida kulusid kokku on veel enamgi.

Töös on oluline ka energia sääst, kuna kasutame raadiosidet ja alati ei pruugi olla püsivaid toiteallikaid siis ka vastavalt sellele tehakse arendusplaadi valik ja kasutatavate protokollide valik, et edastada andmeid võimalikult energiasäästlikult. Selleks vaja ka kasutada sõnumiedastusprotokolle mis väikese koodijalajäljega ja samas turvaline.

Andmete saamiseks kasutame erinevaid andureid mis on ühendatud maketeerimisalusel kaablitega arendusplaadi sisendite ja väljunditega ning saadud andmeid edastatakse minutilise intervalliga pilveplatvormi kus edasi suuname andmed andmebaasi ja sealt omakorda võtame andmed visualiseerimiseks. Tulemuseks saame muutuvad andmed ajas mis on visualiseeritud graafikute ja näidikuna.

Peamised töö ülesanded olid:

- Arendusplaadi riistvara uurimine ja vajalike programmide installeerimine.
- Andurite integreerimine arendusplaadi erinevate sisendite ja väljunditega
- Andmete edastamine raadiosidet kasutades.
- Andmete salvestamine andmebaasis ja visualiseerimine.

1 RAADIOSIDEVÕRK

NB-IoT võrgu valikul on lähtutud eelistest teiste võrkude ees võrreldes tavalise GSM võrguga kus mobiiltelefonid töötavad, siis eeliseks on juba see, et leviala on *NB-IoT* võrgul parem ja samas ka kasutatavad seadmed energiasäästlikumad. Kattuvus on Eestis täielik, võrk on teenusepakkujate (võrguoperaatorite) poolt välja ehitatud. Piisab ainult seadmest mis toetab *NB-IoT* võrku ja vajalik ka SIM kaart mis on kergesti ja odavalt saadav igalt poolt poodidest. Odavaim pakett mis on lepinguline oli 99 senti kuumaksega mida pakuti eraklientidele.

On ka trend, et lisatakse juba *NB-IoT* seadmetele eSIM võimekus. Võrdluseks veel näiteks *LoRaWAN* mis ei ole litsentseeritud, aga kattuvus kahjuks ei ole täielik ja seega on eelistatud *NB-IoT* võrk mis küll on tasuline aga kasutusala kus saab seadet kasutada andmete edastamiseks on suurem. [17]

1.1 NB-IoT

Narrowband-Internet of Things ehk *NB-IoT* on võrgutehnoloogia mis on mõeldud asjade internet jaoks. Annab meile võimaluse suhelda seadmete ja lõpptarbijate vahel väikeseid andmehulkasid edastades ja tarbides vähe energiat. Mõeldud on statsionaarsetele seadmetele sest võrkude ja tugijaamade vaheline liikumine võib probleemiks osutuda. [18]

Kasutusalad kus on hea kasutada *NB-IoT*-d:

- põllumajanduses monitoorimiseks;
- Kütuse-, vee- ja muud torujuhtmete haldamise rakendused;
- Parkimine, jäätmekäitlus, tänavavalgustus ja teised targa linna rakendused;
- Kodu- ja hooneautomaatika rakendused. [18]

Kuna *NB-IoT* levib paremini ka vee all ja keldrites kus tavapärasel GSM ja LTE võrgul levi ei ole siis on võimalik neis keskkondades kasutada andmete edastamiseks just seda tehnoloogiat. [18]

2 PROTOKOLLID

Andmete edastuse protokolliks on valitud *MQTT* mis on mõeldud ka ühtlasi *IoT* seadmete vaheliseks suhtluseks ja mis on ka väikese koodijalajäljega. *MQTT* andmeedastuse põhikomponendiks on *Broker* ehk maakler (vt. Joonis 2.1) ja seega andmeid saavad kõik kliendid kes soovivad ja kes on maakleriga ühenduses. [2]

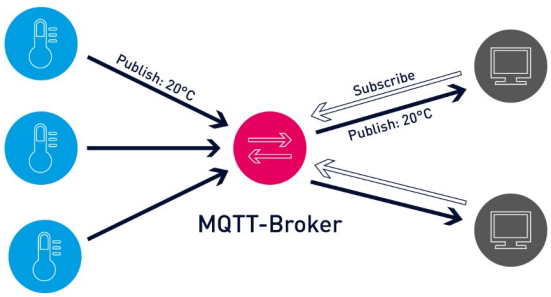
Miks just valitud *MQTT* mitte näiteks *CoAP* protokoll siis selle pärast, et *MQTT* on vägagi paindlik ja toimib puhtalt binaarandmete edastajana. Vastavalt aga *CoAP* protokolliga mis on loodud töötama koos veebi liidesega. *CoAP* on loodud puhverserverite kaudu koos toimima *HTTP* ja *RESTful* veebiga. *CoAP* järgib kliendi/serveri mudelit, kliendid esitavad serveritele päringuid, serverid saadavad vastused tagasi. [16]

Eelistatud oli ka *I2C* kuna on paindlik ja saab kasutada mitme seadme ühendamiseks, samamoodi ka *UART*. *I2C*-l on ka vookontroll ja see teostab andmete valideerimist, seega on andmete terviklikkus ja nende edastamise viis usaldusväärne. Üldiselt on ka *I2C* kiirem kui *UART*. [19]

2.1 MQTT

See on sõnumiedastusprotokoll mis on mõeldud kasutamiseks juhtudel kui on vaja klientidel väikest koodijalajälge ja on ühendatud ebausaldusväärsesse või piiratud ribalaiusega võrku. [2]

MQTT töötab TCP/IP peal, kasutades AVALDA/TELLI (PUSH/SUBSCRIBE) topoloogiat. *MQTT* arhitektuuris on kahte tüüpi süsteeme, kliendid (clients) ja maakler (broker). Maakler on server kellega kliendid suhtlevad. Maakler võtab klientide teated vastu ja saadab need edasi teistele klientidele. Kliendid omavahel ei suhtle vaid suhtlevad läbi maakleri ning iga klient võib olla edastaja, tellia või mõlemad korraga (vt Joonis 2.1). [2]



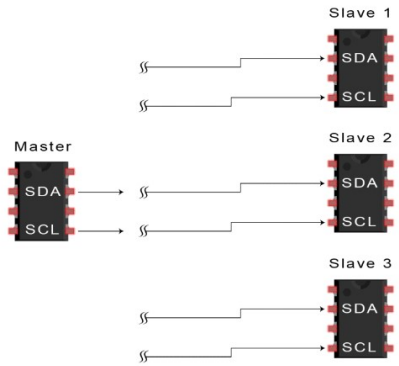
Joonis 2.1 MQTT maakler ja kliendid [2]

Sõnumid avaldatakse teemadena ja teemad on hierarhias olevad struktuurid, mis eraldatakse kaldkriipsuga (/). Struktuur sarnaneb failisüsteemi kataloogipuu

struktuuriga. Struktuur nagu andurid/andur/näit/ võimaldab tellijal määrata, et andmeid tuleks saata ainult klientidelt kes edastavad teemas andurid/andur/näit/. Konkreetsed teemad ei ole *MQTT*-s otseselt loodud, kui näiteks maakler saadab andmeid teema kohta mida ei ole loodud siis luuakse antud teema ja kui kliendid tahavad siis saavad teema tellida. [2]

2.2 I2C

Andmevahetus toimub üle kahe juhtme (vt. Joonis 2.2) *SDA* (andme jadaliin) ja *SCL* (liini taktsagedus). Kuna *I2C* on andmeedastusprotokoll siis andmed edastatakse üle ühe juhtme *SDA* bittide kaupa. Ühendus on sünkroonis, nii väljuvad bitid sünkroniseeritakse vastavalt taktsagedusele mida edastab alati ülem pool (master). [14]



Joonis 2.2 I2C topoloogia [14]

Kiirused mida saab kasutada on järgnevad:

- Standard 100 kbps;
- Kiire 400 kbps;
- Väga kiire 3.4 Mbps;
- Ülikiire 5 Mbps. [14]

Ülemate (master) seadmete arv on piiramatu ja alamate (slave) arv võib olla kuni 1008 seadet. [14]

Andmed edastatakse sõnumitena ja sõnumid omakorda andmeraamideks mis sisaldavad alamseadme (slave) binaaraadressi ja ühte või mitut andmekaadrit. Sõnum sisaldab ka algus ja lõpetamise tingimusi, lugemis/kirjutusbitte ja *ACK/NACK* bitte iga andmekaadri vahel. [14]

3 ARENDUSPLAAT

Arendusplaat nRF9160 on valitud peamiselt oma väikese voolutarbe poolt võrreldes mõne teise tootja arendusplaadiga mis on mõeldud NB-IoT võrku. Allolevast jooniselt on näha ka minimaalsed voolutarbed võrreldes Arduino MKR NB 1500 arendusplaadiga (vt. Joonis 3.1). Omakorda võimaldab väike voolutarve hoida seadet töös korralikku akut kasutades 10 ja enam aastat, mis omakorda teeb andmete kauglugemise suhteliselt hooldusvabaks.

	MKR NB 1500	nRF9160
Mikrokontroller	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	ARM® Cortex® -M33
Ühenduvus	NB-IoT / LTE-M	NB-IoT / LTE-M
	MicroSIM	MicroSIM ja eSIM
	LTE bands 1, 2, 3, 4, 5, 8, 12,	Cat-M1: B1, B2, B3, B4, B5, B8, B12, B13, B14, B18, B19, B20, B25, B26, B28, B66
	13, 18, 19, 20, 25, 26, 28	Cat-NB1/NB2: B1, B2, B3, B4, B5, B8, B12, B13, B17, B19, B20, B25, B26, B28, B66
Voolutarve (LTE M1)	min 100 mA / max 190 mA	min 18 µA
Voolutarve (LTE NB1)	min 60 mA / max 140 mA	min 37 µA
Mälu	256KB Flash, 32KB SRAM	1 MB flash, 256 kB RAM

Joonis 3.1 arendusplaatide parameetrid [20] [21]

Lõputöös on kasutanud arendusplaati *nRF9160 DK* mis on riistvaraline arendusplatvorm, mida kasutatakse rakenduste tarkvarade kujundamiseks ja arendamiseks *LTE-M* ja *NB-IoT* võrgus. Platvorm on avatud lähtekoodiga ja hulgaliste näidis koodidega mida omavahel kombineerides võimalik kasutada erinevate projektide raames.

Arenduskomplekt sisaldab kõiki vajalikke väliseid vooluringe, näiteks abonendi identiteedimoodulit (SIM) kaardi hoidja ja antenn ning annab arendajatele juurdepääsu kõikidele I/O väljaviikudele ja mooduli liidestele (vt Joonis 3.2). Sellel on spetsiaalne *LTE-M* ja *NB-IoT* antenn, mis toetab laia valikut sagedusalasid, et töötada ülemaailmselt. LTE võrkudes B1, B2, B3, B4, B5, B8, B12, B13, B14, B17, B18, B19, B20, B25, B26, B28 ja B66 on sertifitseeritud. [4] [22]

Kõik *GPIO*-d ja liidesed on saadaval pistikute kaudu. Komplekt ühildub Arduino Uno Rev3-ga, mis tähendab, et seda saab hõlpsasti ühendada väliste seadmetega.[4]

Võimalik programeerida LED-e (4), nuppe (2) ja lülitid (2), et lihtsalt kasutada programmis sisendeid ja väljundeid (vt Joonis 3.2). nRF9160 DK-l on nii SIM kaardi pesa kui ka võimekus kasutada eSIM-i mids sisaldub arenduskomplektis.[4]

3.1 Riistvara

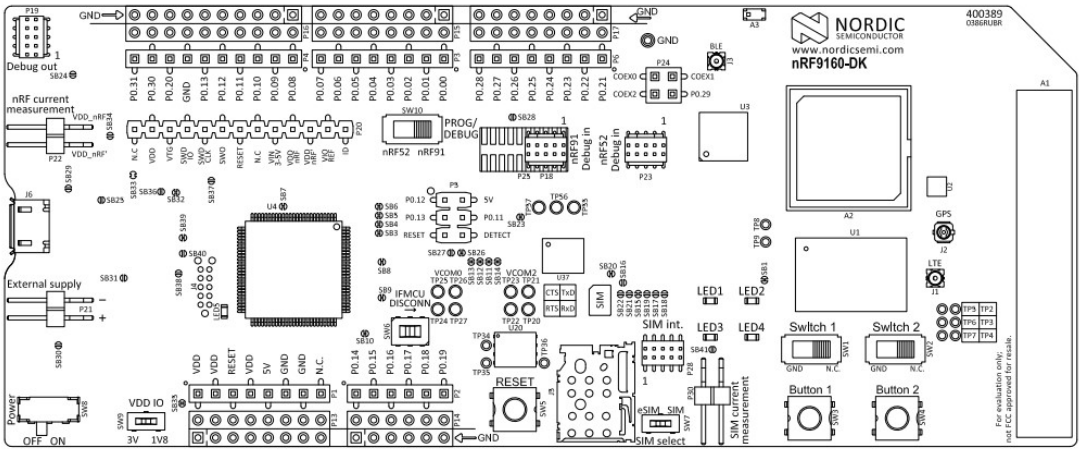
Mikrokontroller:

- ARM Cortex -M33;
- 1 MB mälu;

- 256 kB vähese lekkega RAM;
- ARM Trustzone, riistvaraline turvalaiendustehnoloogia;
- ARM Cryptocell 310, integreeritud turbetuum, mis koosneb nii riist- kui ka tarkvarakomponentidest;
- kuni 4x SPI ülem/alluv (ik. master/slave) koos EasyDMA-ga;
- EasyDMA on hõlpsasti kasutatav otsejuurdepääsu moodul, mida mõned välisseadmed kasutavad RAM-ile otsese juurdepääsu saamiseks;
- kuni 4x I2C-ühilduvusega kahejuhtmeline ulem/alluv (ik. Master/slave) koos EasyDMA-ga;
- I2S koos EasyDMA-ga. I2S on elektrilise jadaliidese standard, mida kasutatakse digitaalsete heliseadmete ühendamiseks;
- digitaalse mikrofoni liides koos EasyDMA-ga;
- 4x impulsi laiuse modulaatori seade koos EasyDMA-ga;
- 3x 32-bitine taimer loenduri režiimiga;
- 2x reaalajas loendur;
- programmeeritav perifeerne ühendus;
- 32 üldotstarbelist I/O kontakti;
- toitepinge: 3,0 – 5,5 V. [4]

I/O väljaviigud ja tähendused (vt Joonis 3.2):

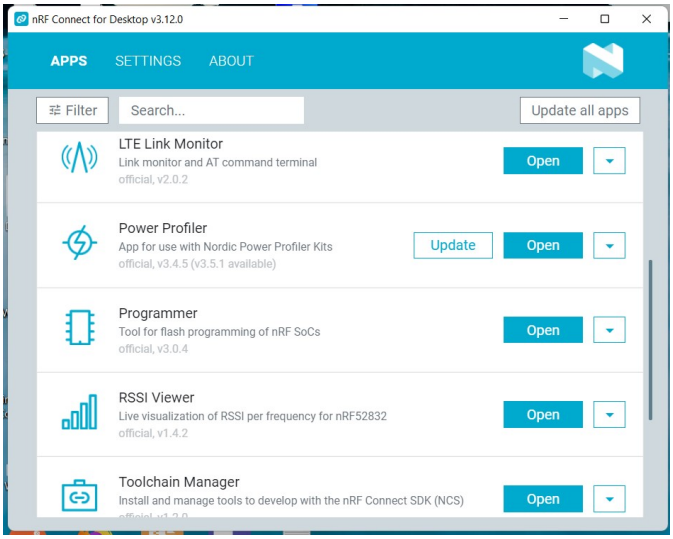
- P0.00, P0.01, P0.14, P0.15 - kasutatakse teise UART-ühendusena MCU liidesega;
- P0.02, P0.03, P0.04, P0.05, P0.06, P0.07, P0.08, P0.09 - vaikimisi ühendatud nuppude, liuglülitite ja LED-idega mis asuvad arendusplaadil;
- P0.17, P0.18, P0.19, P0.21, P0.22, P0.23, P0.24, COEX0, COEX1, COEX2 - kasutatakse nRF9160 ühendamiseks nRF9160 DK plaadiga;
- P0.26, P0.27, P0.28, P0.29 - kasutatakse peamise UART-ühedusena MCU liidesega;
- P0.11, P0.12, P0.13, and P0.25 - kasutatakse välise mälu liidese jaoks;
- P0.06 – katkestab liini I/O laiendustega;
- P0.30 - I2C andmeliin;
- P0.31 – I2C taktsagedus. [4]



Joonis 3.2 arendusplaadi nRF9160 DK pealtvaade [4]

3.2 Tarkvarad ja programmid

Arendusplaadi *nRF9160 DK* rakendus ja modemi tarkvara on eraldi alla laetav Nordic Semiconductor kodulehelt. Plaadiga töötamiseks on vaja programmi *nRF Connect*, mis on nagu tööriistakohver mille sees on vajalikud tööriistad ehk omakorda programmid mis täidavad kindlaid funktsioone (vt Joonis 3.3). Arendusplaadi ja arvutivaheline ühendus toimub läbi USB kaabli, läbi mille saab ka arendusplaat enda toite kui ei kasutata lisatoidet.



Joonis 3.3 nRF Connect

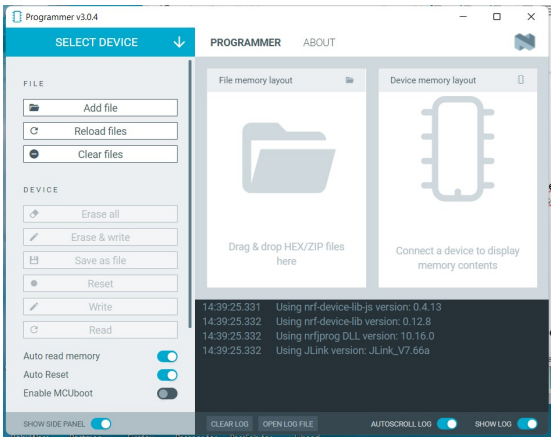
Antud lõputöös on neist kasutatud:

- Programmer;
- LTE Link Monitor;
- Toolchain Manager;

- nRF Connect SDK;
- VS Code.

3.2.1 Programmer

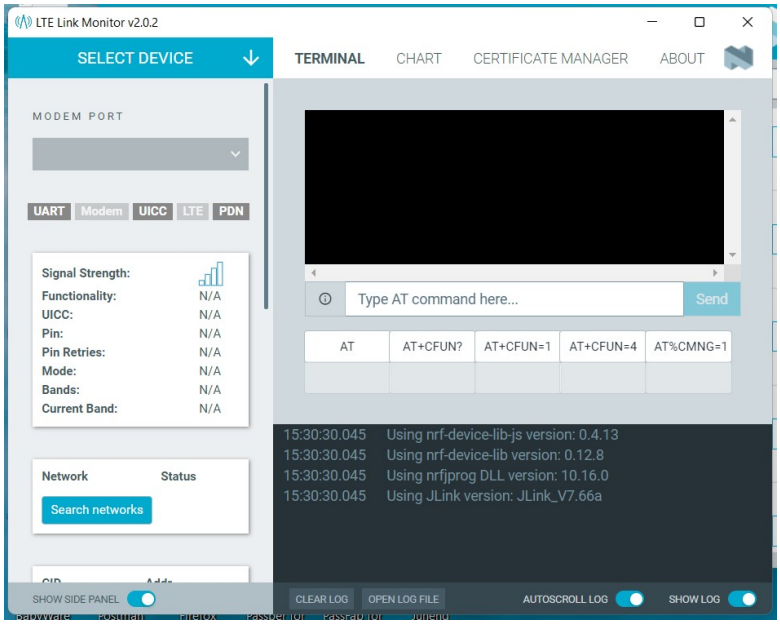
Rakendus võimaldab programmeerida arendusplaadi SoC-d (System on Chip). Saab faile lihtsalt lohistades vastavasse aknasse paigutada ning seadmelt lugeda, kirjutada või kustutada, näitab ka ära kui palju on mälust kasutusel(vt Joonis 3.4). Antud programmiga kirjutatakse ka modemi tarkvara peale mis on alla laetav Nordic Semiconductor kodulehel zip failina. Modemi tarkvara on vajalik eelnevalt peale laadida, et saaks hakata modemit programmeerima ja oleks võimalik kasutada SDK-d.[23]



Joonis 3.4 Programmer

3.2.2 LTE Link Monitor

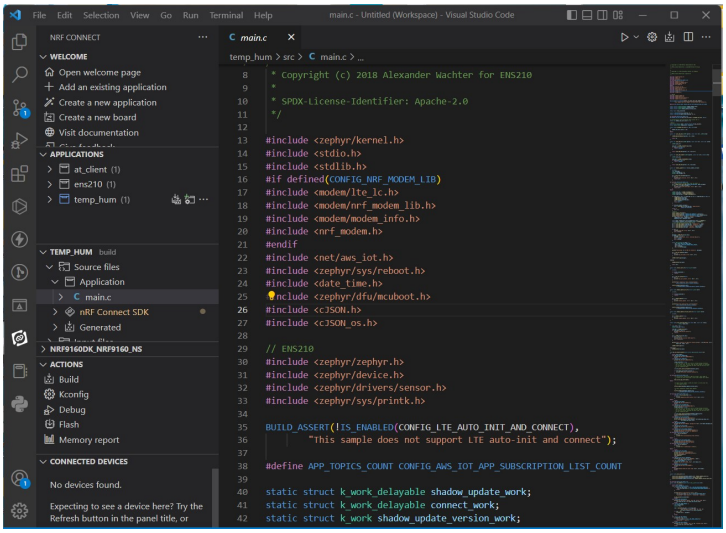
Arendusplaadi nRF91 seeria kasutab püsivara käitamiseks rakenduse tuumas *AT* käske API (Application Programming Interface) modemi juhtimiseks. *LTE Link Monitor* on modemi klientrakendus, mis jälgib modemi ning lingi olekut ja tegevust AT-käskude abil. AT ehk tähelepanu ingliese keelest *attention* käsud on juhised, mida kasutatakse modemi juhtimiseks (vt Joonis 3.5). [23] [24]



Joonis 3.5 LTE Link Monitor

3.2.3 NRF Connect SDK ja Visual Studio Code

Haldamistööriist SDK (software development kit) tööriistade versioonidele Windowsis ja Macis. VS Code-s (Visual Studio Code) saab kasutada vastavaid SDK versioone, mille abil rakendusi arendada ja programmeerida (vt Joonis 3.6).



Joonis 3.6 NRF Connect laienduse näidis Visual Studio Code-s

SDK-d sisaldavad hulgaliselt erinevaid rakenduste malle mida kasutades on võimalik katsetada ja testida erinevaid rakendusi koos arendusplaadi funktsioonide ning võimalustega. Kõikide näidis mallide dokumentatsioon on lahti seletatud Nordic Semiconductor kodulehel. Samuti võimalik ka vaadata erinevaid vebinare kus teatud mallidest ja nendega töötamisest räägitakse.

3.3 Arendusplaadi nRF91 seeria võrgurežiim

Modem toetab *LTE-M* ja *NB-IoT* võrke, vaikimisi käivitub modem *LTE-M* režiimis mis on aga konfigureeritav. LTE Link Control draiveri kasutamisel saab valida *LTE-M* *CONFIG_LTE_NETWORK_MODE_LTE_M* või *NB-IoT* režiimis *CONFIG_LTE_NETWORK_MODE_NBIOT* mis tuleb määrata prj.conf failis (vt. Joonis 3.7). [25]

```
18  # LTE link control
19  CONFIG_LTE_LINK_CONTROL=y
20  CONFIG_LTE_NETWORK_MODE_NBIOT=y
21  CONFIG_LTE_AUTO_INIT_AND_CONNECT=n
22
```

Joonis 3.7 NB-IoT võrgurežiim

NB-IoT režiimi on võimalik minna ka konkreetset draiveri seadistusi programmis kasutamata, selleks tuleb enne modemi käivitamist anda *AT* käsk *AT%XSYSTEMMODE=0,1,0,0* . Reziimi muutmiseks kui modem on juba tõõrežiimil tuleks toimida järgnevalt sisestades *AT* käsud:

- AT+CFUN=4 (lülitab modemi välja);
- AT%XSYSTEMMODE=0,1,0,0 (seadistab võrgurežiimi *NB-IoT* võrgule);
- AT+CFUN=1 (lülitatakse modem töörežiimi). [25]

AT käske on võimalik anda modemile kasutades *LTE Link Monitor* terminali kus saab sisestada käske.

4 ANDURID

Arendusplaadil kasutatavad andurid on erinevate väljunditega, et proovida erinevaid lahendusi. Mõned on üle *I2C* liini nagu *ENS210* ja *VL53L0X,* samuti ka analoog väljundiga milleks on *CO* ehk vingugaasiandur. Kasutuses on ka ultraheliandur *HC-SR04* mis vajab ühte digitaalset sisendit ja väljundit.

4.1 Temperatuuri- ja niiskuseandur ENS210

Temperatuuri ja niiskuse mõõtmiseks on kasutatud *Temp&Hum 6 Click* (vt Joonis 4.1) mille põhikomponent on *ENS210* mis mõõdab temperatuuri ja suhtelist õhuniiskust väga täpselt. Andur on *I2C* liidesega ja töötab pingega 3.3V toodab seda firma AMS AG. *ENS210* sisaldab suure täpsusega soojusandurit, mis suudab mõõta temperatuuri vahemikus -40°C kuni 100°C ning säilitab täpsuse ±0,5°C. Kui seda kasutatakse vahemikus 0 °C kuni 70 °C, tüüpiline täpsus on ±0,2°C. Seda saab kasutada pikka aega, kuna sellel mõõte täpsus võib muutuda ainult 0,005°C aastas. [26]

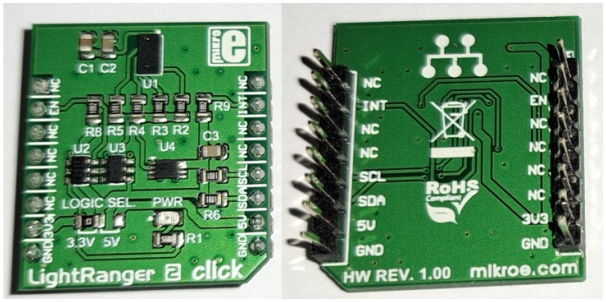
Niiskusandur on kondensaatoripõhine andur, mis koosneb niiskustundlikust suure pindalaga kondensaatorist. Niiskustundlik kiht võimaldab mahtuvust muuta proportsionaalselt suhtelise õhuniiskusega. Mahtuvus on lineaarne sõltuvus temperatuurist, mis tagab suure täpsuse. Anduri täpsus varieerub vahemikus ±2,5% - ±5,5%, olenevalt mõõtmistingimustest. [26]

Joonis 4.1 Temp&Hum 6 click

4.2 Kauguse ja liigutuste tuvastamise andur VL53L0X (ToF)

VL53L0X on Time-of-Flight ehk *ToF* andur, mis määrab kauguse laseriga, kogu andur asub väikeses plastik kestas. Andur suudab mõõta absoluutseid vahemaid kuni 2 meetrit. Andmevahetus käib üle *I2C* liidese mis on plaadil märgitud *SCL* ja *SDA*, tööpinge on 3.3V. Tootjafirma MikroElektronika on disaininud selle anduri trükkplaadile LightRanger 2 click, mida on siis kerge kasutada teiste arendusplaatidega, kuna seal

on kõik vajalikud väljaviigud välja toodud, et kasutada maketeerimislaual (vt Joonis 4.2). [13]

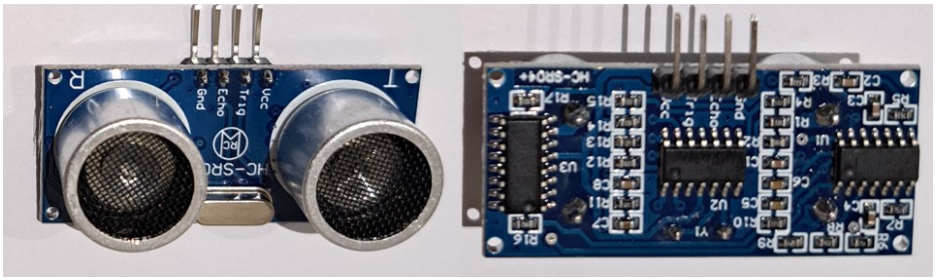


Joonis 4.2 LightRanger 2 click

4.3 Ultraheli moodul HC-SR04

Andur kasutab kauguse mõõtmiseks ja objektide tuvastamiseks ultraheli, ning mille mõõtetäpsus võib ulatuda kuni 3 millimeetrini. Tööraadius jääb vahemiku 2 sentimeetrit kuni 400 sentimeetrit, ning mõõtenurk on 15 kraadi. Andur koosneb ultrahelisaatjast ja vastuvõtjast ning juhtahelatest mis on teiselpool trükkplaati (vt. Joonis 4.3). Tööpinge on 5 V ja üks sisend millele andes signaali genereeritakse 40 kHz helisagedust (trig) ja väljund jalg mis tuvastab saadud helisageduse kui on toimunud kuskilt objektilt peegeldus (echo). [27]

Anduri tööpõhimõte seisneb selles, et saadetakse välja 10 mikrosekundi jooksul kaheksa 40 kHz signaali impulssi mis objektilt peegeldades vastuvõtjasse annab pinge echo jalale ja kui signaal tuleb uuesti eco jalale siis pinge on uuesti 0V. Kauguse mõõtmiseks kasutatakse aega mille vältel on echo jalal pinge, mis siis jagatakse kahega kuna heli saabumine vastuvõtjasse on edasi ja tagasi levinud signaali aeg. Helilevimise kiiruseks tubastes tingimustes loetakse umbes 340 m/s.



Joonis 4.3 HC-SR04

4.4 Vingugaasiandur MQ-7

Andurit on mugav kasutada maketeerimislaual ja on kompaktne, tuvastab süsinikmonooksiidi olemasolu. Anduri elemendiks on kasutatud *MQ-7* vingugaasiandur mis on arendatud Zhengzhou Winsen Electronics Technology poolt. Anduri gaasitundlik

kiht on valmistatud tinaoksiidist (SnO2), mille juhtivus puhtas õhus on madalam ja juhtivus suureneb süsinikdioksiidi taseme tõusuga. [28]

Joonis 4.4 CO click

Andur on paigutatud trükkplaadile (vt. Joonis 4.4) mida saab kasutada maketeerimislaual mugavasti, kuna kõik väljaviigud on olemas. Tööpinge on anduril 5V ja signaal saadakse analoogväljundist. Potensiomeeteriga saab reguleerida anduri tundlikust. [28]

5 AWS (AMAZON WEB SERVICES)

Pilve platvorme on mis on mõeldud arendajatele oma rakenduste loomiseks ja arendamiseks on mitmeid, nendest tuntumad on näiteks Amazon Web Services kui ka Azure Cloud Service. Iseenesest neil kahel midagi väga suurt erinevust ei ole Azure Cloud Service teenust pakub Microsoft ja AWS teenust pakub Amazon. Allolevas joonisel on näha mõned erinevused keskkondade vahel (vt. Joonis 5.1). [29]

	Azure	AWS
1	Kasutatakse arvutamiseks virtuaalseid masinaid	AWS-is kasutatakse arvutamiseks Elastset Arvutuse Pilve
2	Kasutab salvestamiseks plokke	Kasutab salvestamiseks lihtsat salvestusteenust
3	Azure on virtuaalne võrk	AWS on virtuaalne privaatpilv
4	Saab rakendusi hõlpsasti arendada	Rakendusi on veidike keerulisem arendada
5	Kasutatakse SQL-i andmebaase, MySQL-i, Cosmos DB-d jne.	Andmebaasidena kasutatakse RDS-i ja DynamoDB-d
6	Kallim kui AWS	Hind on madalam kui Azure-l
7	On hea platvorm	On parem platvorm
8	Kasutatakse maailmas vähem	Kasutatakse maailmas rohkem

Joonis 5.1 Azure ja AWS võrdlus [29]

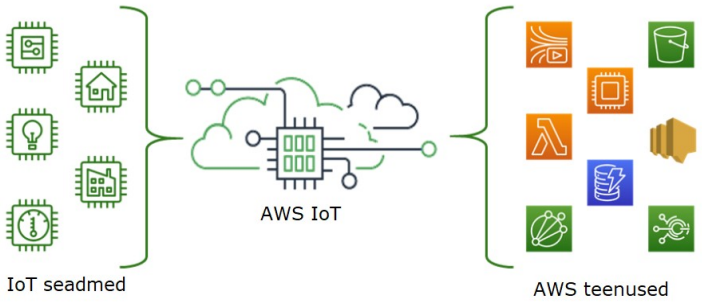
Valituks osutus AWS keskkond, kuna on odavam, kasutatakse rohkem ja võrdlusest ka märgitud, et parem platvorm. Varasem töökogemus antud keskkonnas puudus ja oli soov proovida keskkonna erinevaid võimalusi andmete vastuvõtmiseks kuni nende visualiseerimiseni välja.

Antud lõputöös on autor kasutanud AWS-is järgmisi platvorme:

- AWS IoT Core (maakler)
- Amazon Timestream (klient)
- Amazon Grafana (rakendus)

Kus keskkonnaandmeid edastab arendusplaat nRF9160 DK (klient).

AWS IoT on pilveteenus, mis aitab ühendada *IoT* seadmed teiste seadmete ja AWS-i pilveteenustega. AWS IoT pakub seadme tarkvara, mis aitab integreerida *IoT* seadmeid AWS IoT põhistesse lahendustesse (vt Joonis 5.2). [30]



Joonis 5.2 AWS IoT [30]

AWS IoT Core on maakler mis toetab seadmeid ja kliente, mis kasutavad sõnumite avaldamiseks ja tellimiseks *MQTT*,WSS ja samuti HTTPS protokolli. [30]

Amazon Timestream on kiire, skaleeritav, täielikult hallatav, eesmärgipäraselt loodud aegridade andmebaas, millega on lihtne salvestada ja analüüsida triljoneid andmehulkasid mis on saadetud päeva jooksul. [30]

Amazon Grafana on andmete visualiseerimise teenus millega saab luua isoleeritud Grafana serverid mida nimetatakse tööruumideks inglise keeles workspaces ilma, et peaks enda arvutisse Grafana paigaldama. [30]

6 TEHTUD KATSED JA TULEMUSED ANDURIGA ENS210

Arendusplaadi *nRF9160 DK* dokumentatsiooni uurides oli proovitud erinevaid variante kuidas andmeid saata ja visualiseerida mille tulemusena sai valitud SDK versiooniga 2.1.0 milles oli koodi mall aws_iot mis oli põhi mida autor kasutas (vt Lisa 1).

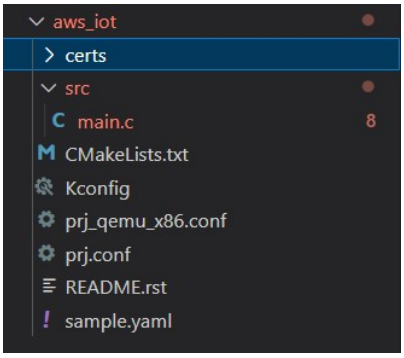
Kuna algselt proovitud meetodiga kus prooviti kasutada kolmanda osapoole teeki (library) ei andnud kahjuks tulemust, kuna see tähendas ka algsete konfiguratsioonifailide põhjaliku/osalist muutmist mida Nordic Semiconductori tehniline tugi ei soovitanud proovida, kuna tulemus poleks tõenäoliselt tööle hakanud. [9]

Projekt aws_iot sisaldas mitmeid faile mida kompileerides saadi rakendus ja mis omakorda sai laadida arendusplaadile ja testida kuidas töötab vastavate tööriistadega mida sisaldas programm nRF Connect.

Põhiprogramm ehk main.c (vt Lisa 1) on programmi üks osa, mis koosnes veel mitmest olulisest failist (vt Joonis 6.1) mis rakenduse ehitamiseks on vajalikud.

Vajalikud failid:

- main.c
- CMakeList.txt
- Kconfig
- Prj.conf

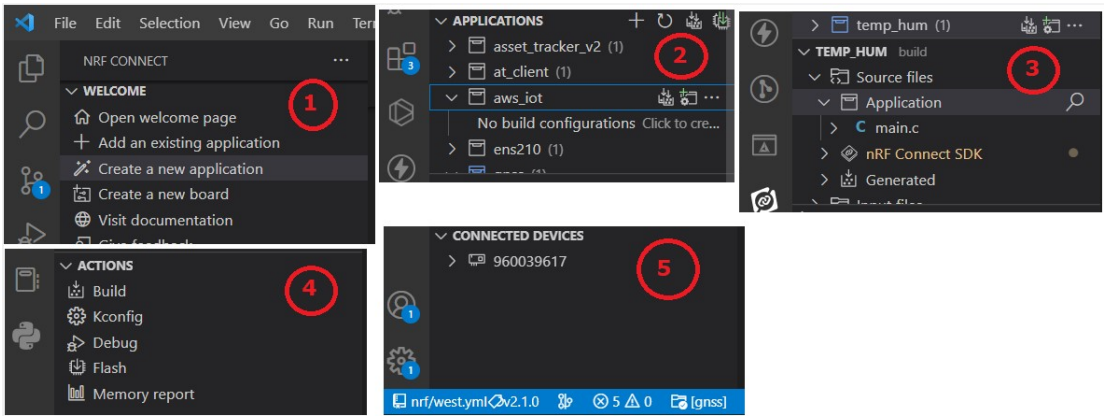


Joonis 6.1 aws_iot malli failid Visual Studio Code-s

Rakenduse ehitamiseks Visual Studio Code abiga pidi kasutama NRF Connect laiendust mis võimaldas rakenduse koodist ja erinevatest failidest valmis ehitada ehk kombileerida. Rakenduse ehitamiseks olulisemaid faile on CMakeList.txt (nõuab laiendust CMake) kus on ära toodud kuidas hakatakse programmi ehitama ja mis teid kasutatakse selleks.

Nordic Semiconductor kasutab oma rakenduste ehitamiseks *Zephyr*-it mis on väike reaalajas toimiv operatsioonisüsteem (RTOS), mis on optimeeritud piiratud ressursiga seadmetele ja mis on välja antud Apache License 2.0 alusel. [31]

Visual Studio Code vasakpoolses akendes kui laiendus NRF Connect on avatud on näha mitmeid erinevaid aknaid (vt Joonis 6.2). Esimeses aknas saab luua rakenduse, vastavalt SDK versioonile on erinevad mallid. Teises aknas on rakendused mis on loodud või vajavad loomist. Kolmandas aknas on loodud rakendus koos failidega mida võimalik katsetuste käigus muuta. Neljandas aknas saab valida toiminguid kui koodis tehakse muudatusi siis vastavate muudatuste alusel rakendus uuesti ehitada või kirjutada programm plaadile. Viies aken näitab ära millise paadiga oled ühenduses.



Joonis 6.2 Visual Studio Code laienduse NRF Connect akende sisu

6.1 Arendusplaadi ja anduri sidumine

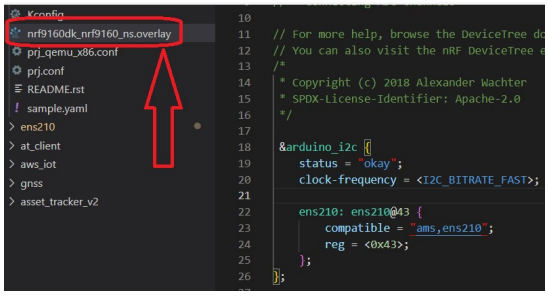
SDK versioonis 2.1.0 oli olemas draiverid anduri ENS210 jaoks, seega sai kasutatud neid, et need kaks koodi omavahel siduda (vt Lisa 1 ja Lisa2). Autor hakkab kasutama malli põhja mis on näidatud Lisa 1, millele lisame juurde Lisa 2 vajalikud osad. Esmalt muudatused prj.conf failis kus lülitatakse sisse vastavad võimalused seda andurit kasutada tarkvaraliselt milleks tuli lisada faili juurde read (vt Joonis 6.3).

```
75
76  # ENS210
77  CONFIG_I2C=y
78  CONFIG_SENSOR=y
```

Joonis 6.3 prj.conf

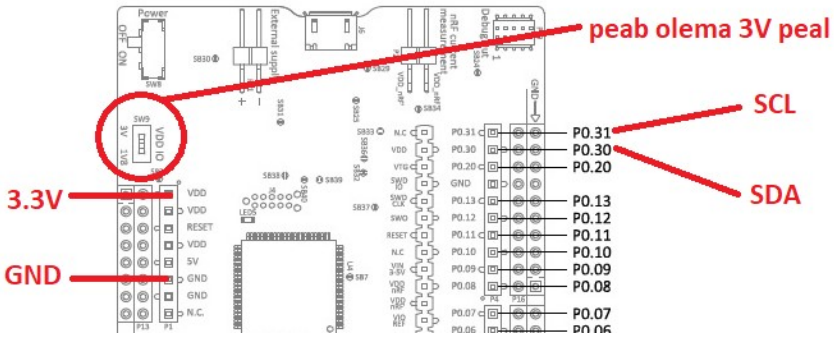
CONFIG_I2C=y näitab, et lülitatakse sisse *I2C* suhtlus protokoll ja *CONFIG_SENSOR=y* siis anduri.

Riistvara jaoks tuli tekitada *nrf9160dk_nrf9160_ns.overlay* fail mis kirjeldas arendusplaadile riistvaralise osa ära (vt Joonis 6.4).



Joonis 6.4 overlay fail

Failis on kirjeldatud staatus mis näitab, et andur on töös, ühenduse kiirus, anduri nimi ja anduri (slave) aadress *I2C* liinil. Andur on ühendati vastavalt nRF9160DK raudvara dokumentatsioonist lähtuvalt (vt joonis 6.5).



Joonis 6.5 nRF9160DK väljaviigud [4]

Hiljem on vaja muuta ka *main.c* failis koodi ennast kus hakatakse edastama anduri näite AWS IoT Core loodud objektile.

6.2 Keskkonnaandmete edastamine AWS IoT Core

Andmete edastamiseks kasutatakse SIM kaarti, eelnevalt vaja PIN koodi küsimine mobiiliga maha võtta kui operaatori poolt ei ole tehtud seda. SIM kaart tuleb sisestada ennem kui arendusplaat on USB kaabliga ühendatud või kui kasutatakse lisatoidet siis see lahti ühendada eelnevalt.

Nimekirjast (vt Joonis 6.1) on vaja muuta järgnevate failide sisu (main.c, Kconfig ja Prj.conf) vastavalt AWS-is registreeritud kontole. Esmalt on vaja kontot AWS keskkonnas seejärel peab looma sinna objekti ja vastava poliitika kirjutama objekti jaoks. Samuti saab peale objekti loomist saadakse sertifikaadid mis on vaja ühenduse loomiseks AWS platvormiga.

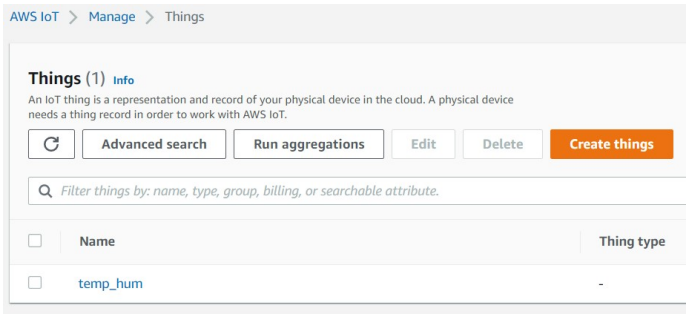
Peale konto loomist tuleb ka valida vastav regioon kus *Timestream* andmebaas on olemas, kuna kõikides pakutavates regioonides ei ole, autoril on valitud *US West Oregon* (us-west-2). Kõik järgnev tuleb luua sinna alla, esmalt loome poliitika (polici) mille seome meie objektiga. AWS IoT core lehel on olemas selline koht menüüs nagu

Security mille alt leiame koha *polici*. Peale poliitika loomist (vt Joonis 6.6) saab luua objekti (Things) (vt Joonis 6.7) mille seome loodud poliitikaga.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

Joonis 6.6 objekti poliitika JSON formaadis

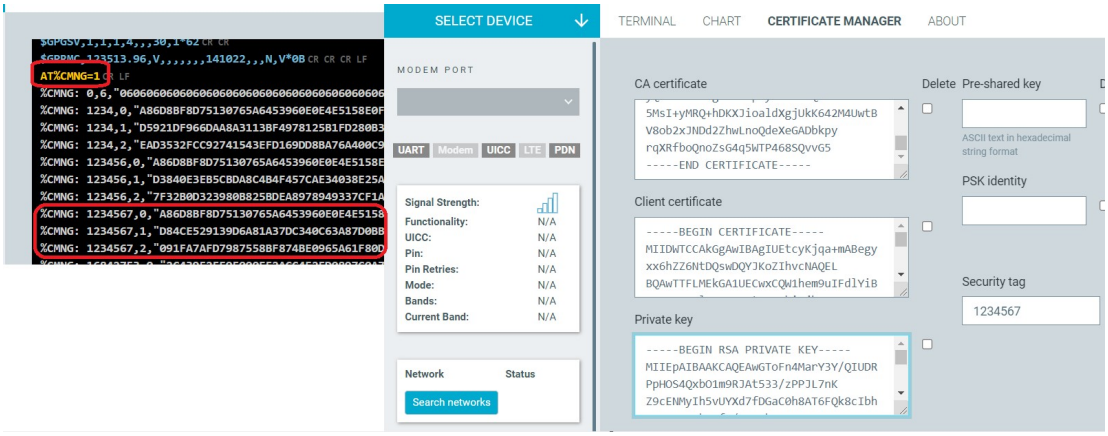
Kui objekt on loodud (Things) ja saadud sertifikaadid siis tuleb need salvestada kohe, kuna hiljem ei ole võimalik neid saada antud objekti kohta. Antud töös on objekt nimetatud *temp_hum* (vt Joonis 6.7). Ülesse peame leidma veel *Settings* lehelt *Endpoint* ja salvestama seal oleva aadressi, mida kasutame hiljem programmi koostamisel.



Joonis 6.7 objekt (Things) temp_hum

Sertifikaadid saab modemisse ülesse laadida vastava SDK versiooni 2.1.0 *at_client* malli järgi. Visual Studio Code-s tuleb luua rakendus *at_client* sellest rakendus luua ja plaadile ülesse laadid. Saadud sertifikaadid on võimalik tekstiredaktoriga avada ning sealt sisu kopeerida. Kui rakendus töötab arendusplaadil saab programmiga *LTE Link Monitor* (vt. Joonis 3.5) AWS sertifikaadid ülesse laadida.

Sertifikaatide laadimisel peab jälgima ja meeles hoidma mälupesanumbri kuhu need laetakse antud töös on vallitud mälupesanumbriks 1234567 (vt Joonis 6.8). Modem ise peab olema võrguühenduseta sellel hetkel kui sertifikaate kirjutatakse mällu, selleks on *AT* käsk *AT+CFUN=4* mis tuleb terminalis kirjutada.



Joonis 6.8 sertifikaatide lisamine modemi

Peale laadimist on võimalik terminalist *AT* käsuga *AT%CMNG=1* kontrollida millises mälupesas on sertifikaadid kirjutatud.

Kui sertifikaadid olemas modemis siis naaseme rakenduse *aws_iot* juurde tagasi ja seal on vaja muuta seadistusi failis prj.conf kuhu lisame loodud sertifikaatide mälupesanumbri ja AWS IoT core *Endpoint,* samuti ka loodud objekt (vt Joonis 6.9).

```
30  # AWS IoT library
31  CONFIG_AWS_IOT=y
32  CONFIG_AWS_IOT_CLIENT_ID_STATIC="temp_hum"
33  CONFIG_AWS_IOT_BROKER_HOST_NAME="a3cptt75zyos7t-ats.iot.us-west-2.amazonaws.com"
34  CONFIG_AWS_IOT_SEC_TAG=1234567
```

Joonis 6.9 prj.conf AWS IoT muudatused

Avades *main.c* kus asub põhi kood on vaja lisada ka ENS210 anduri osa juurde (vt Lisa 2). See kood on vaja võtta osadeks ja osa veidike ümber muuta, selleks kopeerime algselt kogu koodi sisu *aws_iot* koodi lõppu, kus lisame meile vajalikud osad õigetesse kohtadesse.

Alustame koodi ülevalt ja lisame sinna anduri header ehk päise failide viited (vt Joonis 6.10) joonislet on näha rea number kuhu on lisatud.

```
26  #include <cJSON.h>
27  #include <cJSON_os.h>
28
29  // ENS210
30  #include <zephyr/zephyr.h>
31  #include <zephyr/device.h>
32  #include <zephyr/drivers/sensor.h>
33  #include <zephyr/sys/printk.h>
34
35  BUILD_ASSERT(!IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT),
36               "This sample does not support LTE auto-init and connect");
37
```

Joonis 6.10 header failide viited

Koodi algusesse vaja lisada anduri osa, kus seadistame selle väärtused (vt Joonis 6.11), samuti koodi *void main(void)* osasse *koodi osa* kui meie anduriga peaks midagi juhtuma, et veateade väljastataks (vt Joonis 6.11). Anduri näitude saatmiseks leiame

koodi osa kus formuleeritakse *JSON* formaat ja muudame antud koodi anduri andmetega ümber.

```
136     //ENS210
137     sensor_sample_fetch(dev);
138     sensor_channel_get(dev, SENSOR_CHAN_HUMIDITY, &humidity);
139     sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temperature);
140     printk("Temperature: %d.%06d C; Humidity: %d.%06d%%\n",
141         temperature.val1, temperature.val2,
142         humidity.val1, humidity.val2);
143
144     float temp= (temperature.val1+(temperature.val2 * 0.000001));
145     float humid= (humidity.val1+(humidity.val2 * 0.000001));
146     //väärtused JSON formaadis
147     err += json_add_number(reported_obj, "temperature", temp);
148     err += json_add_number(reported_obj, "humidity", humid);
149     err += json_add_obj(state_obj, "values", reported_obj);
150     err += json_add_obj(root_obj, "temp_hum", state_obj);

519 void main(void)
520 {
521     int err;
522
523     printk("The AWS IoT sample started, version: %s\n", CONFIG_APP_VER
524
525     cJSON_Init();
526     //ENS210 start
527     if (!device_is_ready(dev)) {
528         printk("Device %s is not ready\n", dev->name);
529         return;
530     }
531
532     printk("device is %p, name is %s\n", dev, dev->name);
533     //ENS210 end
```

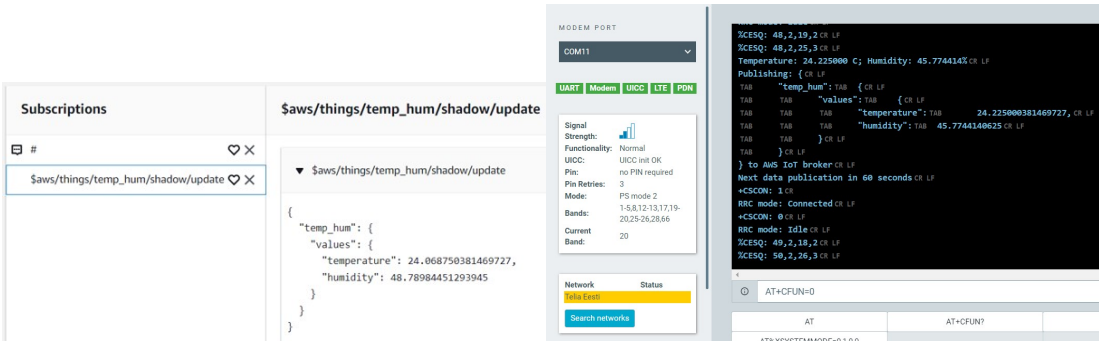
Joonis 6.11 anduri JSON formaat ja vea väljastamine

Vastavalt *sensor.h* päise failile saadakse temperatuudi näit kahes osas (vt. Joonis 6.12).Näit saadakse kahest väärtusest kokku, *val1* mis on täisarv ja *val2* mis on murdarv, aga väärtus väljastatakse täisarvuna. Muutujad *temp* ja *humid* saadakse nende kahe osa liites millest *val2* on eelnevalt korrutatud läbi selliselt, et tulemus oleks murdarv.

```
 *
 *      0.5: val1 =  0, val2 =  500000
 *     -0.5: val1 =  0, val2 = -500000
 *     -1.0: val1 = -1, val2 =  0
 *     -1.5: val1 = -1, val2 = -500000
 */
Kumar Gala, 2 years ago | 2 authors (Anas Nashif and others)
struct sensor_value {
        /** Integer part of the value. */
        int32_t val1;
        /** Fractional part of the value (in one-millionth parts). */
        int32_t val2;
};      Anas Nashif, 3 years ago • cleanup: include/: move sensor.h to
```

Joonis 6.12 val1 ja val2 suurused

Rakenduse loomiseks arendusplaadi jaoks on vaja rakendus ehitada kasutades *Pristine Build,* mille järel rakendus ehitatakse uuesti ja täiendatakse tehtud muudatustega. Seejärel saab kirjutada plaadile kasutades *Erase And Flash To Board.*

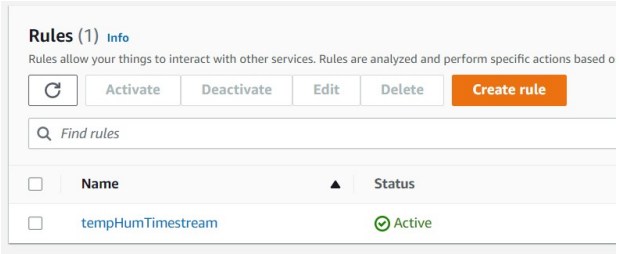


Joonis 6.13 MQTT test client ja LTE Link Monitor

Kui rakendus hakkab plaadil tööle edastatakse väärtused nii *LTE Link Monitor* terminali aknas kui kasutatakse USB kaabliga ühendust kui ka *AWS IoT Core* vastava teema all kasutades *MQTT test client* lehte(vt Joonis 6.13).

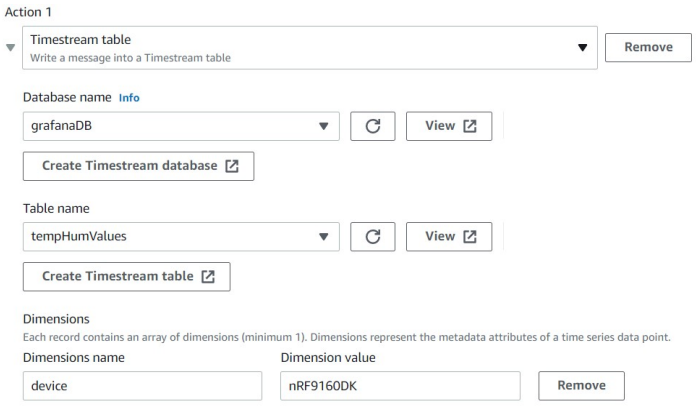
6.3 Andmete salvestamine Amazon Timestream andmebaasis

Andmete edastamiseks Timestream andmebaasi on vaja koostada reegel, mis määrab ära mis teated edastatakse andmebaasi. Menüü lehel *Manage* mille alt edasi liikudes *Message Routing* leiame koha *Rules.* Sinna luuakse reegel (vt Joonis 6.14) ja reeglit tehes kohe ka andmebaas millega see reegel seotakse.



Joonis 6.14 reegel andmete edastamiseks

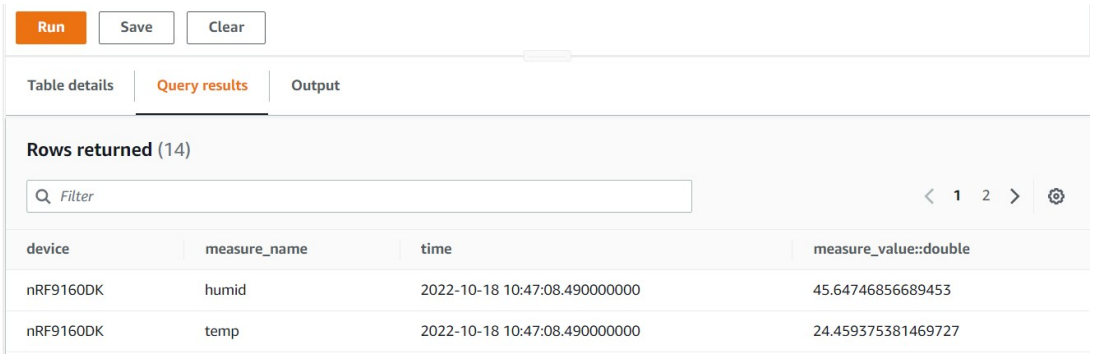
Reegli sisuks kirjutatakse SQL päring mis hakkab andmebaasi andmeid saatma. *SELECT temp_hum.values.temperature as temp, temp_hum.values.humidity as humid FROM '$aws/things/temp_hum/shadow/update'.* Antud päring hakkab andmeid võtma vastavalt saadetud infole mida näitas *MQTT test client* (vt.Joonis 6.13). Edasi luuakse ka andmebaas ja table kuhu andmed saadetakse (vt Joonis 6.15).



Joonis 6.15 reegli sidumine andmebaasiga

Tabelit koostades peab märkima esimese tulba nime *Dimensions name* ja sisu *Dimensions value* mis tulpa kirjutatakse. Tulba sisuks võib ka olla muutuv väärtus mis ei kordu, antud juhul on arendusplaadi nimetus.

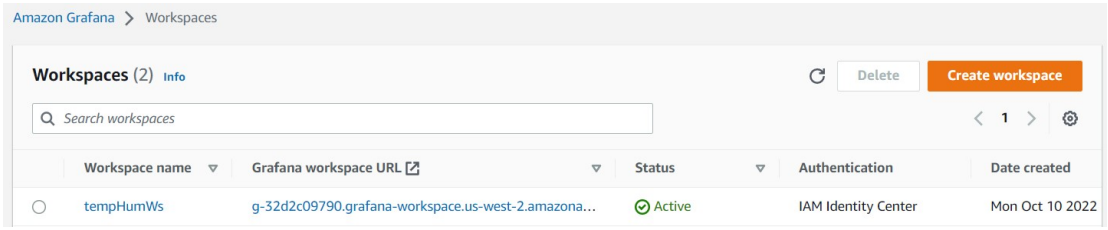
Amazon Timestream lehel olles valides andmebaas ja teha seal *SQL* päring *SELECT * FROM "grafanaDB"."tempHumValues" ORDER BY time DESC*, ilmuvad andmed ajaliselt kahanevas järjekorras mis on salvestatud andmebaasi järgides eelnevat reeglit (vt Joonis 6.16). Andmete edastuse intervall oli jäetud koodis 60 sekundit.



Joonis 6.16 SQL päring Timestream andmebaasist

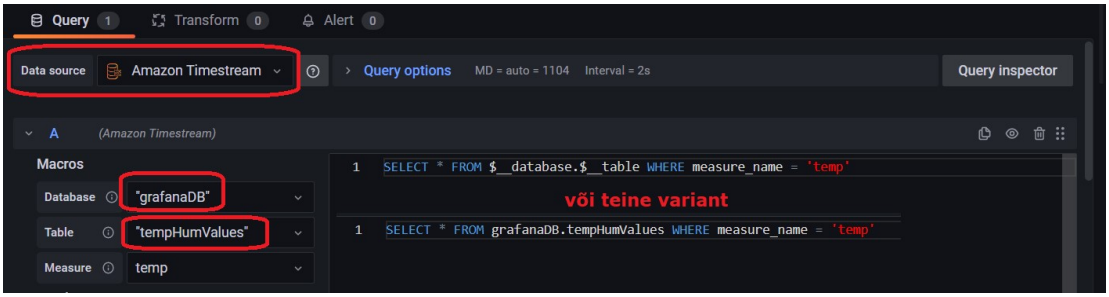
6.4 Andmete visualiseerimine

Eelnevas punktis saadud andmed on võimalik *Grafanaga* visualiseerida, Amazonil on olemas liides selle jaoks, mispärast ei ole vaja installeerida Grafanat enda arvuti. Avades Amazon Grafana on võimalik luua sinna eraldi töölaud, mis annab lingi *Grafana* lehele (vt Joonis 6.17).



Joonis 6.17 Amazon Grafana töölaud

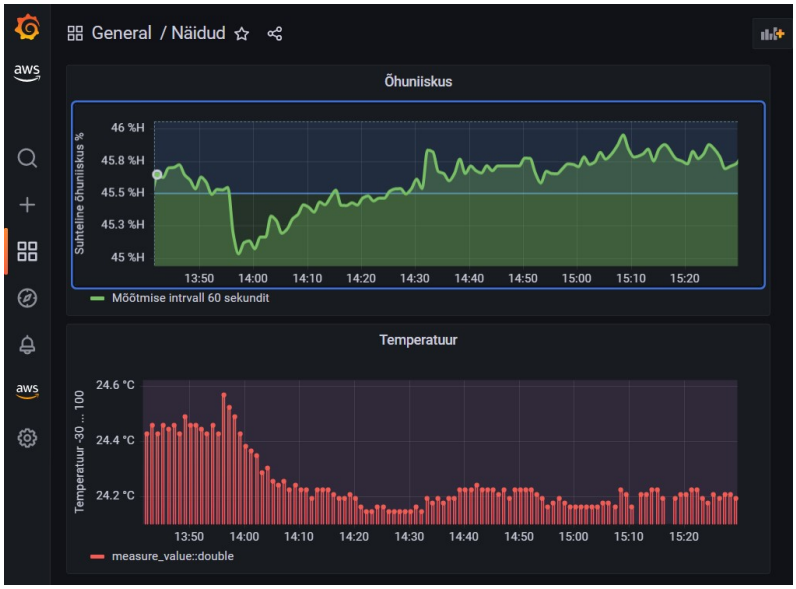
Saadud töölaua lingilt avaneb *Grafana* pealeht koos erinevate valikutega, seal tuleb luua töölaud ja töölauale on võimalik luua erinevaid graafikuid ja näidikuid. Amazon Grafanal on juba ka andmeallikas valitud milleks on Timestream andmebaas ja ei pea eraldi valima kui töölauda luuakse ja sinna näidikuid või graafikuid tekitatakse. Kui töölaud on loodud ja sinna ka graafiku paneel lisatud siis peab üle vaatama, et oleks õige andmebaas ja tabel valitud (vt. Joonis 6.18).



Joonis 6.18 andmeallika valimine

Grafanaga saab samamoodi andmed kätte *SQL* päringuga, kus andmebaasi ja tabeli poole pöördumiseks on kaks viisi (vt. Joonis 6.18). Vastavat päringut tehakse nii mitu korda kui Grafanas määratakse kas siis iga viie sekundi järel või kord päevas näiteks. Intervalli saab muuta töölaual vastavalt vajadusele. Graafiku perioodi saab valida samamoodi kas näiteks viimased 5 minutit või terve aasta vastavalt vajadusele ja rohkemgi.

Graafikut saab kujundada vastavalt soovile, kõiki nimetusi on võimalik redigeerida, graafiku stiili muuta, märkida väärtuste ülempiiri ja alampiiri mida soovitakse kuvada. Kahe erineva graafiku pealt ka hea vaadata sõltuvust üksteisest (vt. Joonis 6.19), kui üks parameeter muutub siis mis juhtub teisega.



Joonis 6.19 temperatuuri ja õhuniiskuse graafik ajas

Õhuniiskuse graafikus on näha joon vahelt läbi, mida on võimalik mistahes tasemele tõsta, et oleks hõlpsam vaadelda näiteks kriitilisemat või üle normi minevat näitu. Samuti saab huvipakkuvama osa graafikust suuremaks teha, et oleks parem vaade väiksema aja ulatuses.

7 ANDURITE HC-SR04, VL53L0X JA MQ-7 INTEGREERIMINE

Uute andurite integreerimiseks tuli kõigepealt iga andur eraldi arendusplaadiga töökorda seada, et olla veendunud anduri töös. Peale mille kontrollimist sai lisatud anduri kood AWS-i koodiga kokku (vt. Lisa 6) ja andmeid hakati saatma samamoodi nagu eelnevalt temperatuuri ja niiskusanduri andmeid lisades *JSON* formaadile uued väärtused juurde.

Iga lisatud anduri kohta on eelnevalt välja töötatud individuaalne kood arendusplaadi jaoks (vt. Lisa 3, Lisa 4 ja Lisa 5). Kui antud kood arendusplaadile laadida ja eelnevalt prj.conf ja ka mõnel juhul overlay failis vastavad muudatused sisse viia siis saab katsetada andurit individuaalselt. Lõplik tulemus on näha (vt. Lisa 6) koodis kus on siis eelnevalt arendatud kood lisatud AWS koodiga kokku, et toimuks vastavate andurite andmete edastus ja oleks võimalik neid visualiseerida.

Failis prj.conf on sisse viidud muudatused (vt. Joonis 7.1) selleks, et lubada kasutada koodis analoog-digitaal konverterit mis on vajalik vingugaasianduri jaoks ja *GPIO*-d, et saaks sisendeid ja väljundeid programmeerida. Samamoodi pidi lisama ka overlay faili muudatuse *VL53L0X* anduri jaoks mis suhtles üle *I2C* liini. Kuna antud anduri draiverid olid olemas SDK-s siis ühilduvuseks tuli vastav andur valida ja määrata aadress ära mis oli leitav samamoodi daraiverite failide alt.

Joonis 7.1 prj.conf ja overlay failid

Muidugi tuli teha ka muudatusi AWS-is kuna seal uute andurite andmete edasi saatmiseks puudus reeglis päring nende andmete kohta. Sai lisatud päringusse ka nende andmete päring juurde sarnaselt ens210 omaga. Samamoodi sai lisatud Grafanas vastavad näidikupaneelid juurde ja seadistatud sarnaselt nagu eelenevate näidikute paneelid (vt. Lisa 7).

Koodi ülaosas lisame vastavad päise failid (laiendusega .h) mida koodi kompileerimiseks on vaja (vt. Joonis 7.2). Defineeritakse led0 ja led1 ja analoog digitaal konverteri parameetrid, ning sisendkanali parameetrid.

```
#include <zephyr/drivers/gpio.h>
//CO
#include <drivers/adc.h>
#include <string.h>
#include <math.h>
#include <hal/nrf_saadc.h>
//HC-SR04
#define LED0_NODE DT_ALIAS(led0)
#define LED1_NODE DT_ALIAS(led1)
//CO
#define ADC_DEVICE_NODE     DT_INST(0, nordic_nrf_saadc)
#define ADC_RESOLUTION      10
#define ADC_GAIN        ADC_GAIN_1_5
#define ADC_REFERENCE       ADC_REF_INTERNAL
#define ADC_ACQUISITION_TIME    ADC_ACQ_TIME(ADC_ACQ_TIME_MICROSECONDS, 10)
#define ADC_CHANNEL_ID  0
#define ADC_CHANNEL_INPUT   NRF_SAADC_INPUT_AIN1
#define BUFFER_SIZE 1
```

Joonis 7.2 parameetrid ADC, A0, led0 ja led1 jaoks.[32]

Analoog digitaal konverter seadistatakse ära (vt. Joonis 7.3) kasutades eelnevalt defineeritud väärtusi. Kasutame 10 bitist konverterit sisendis A0 ehk väljaviik P0.14 (vt. Joonis 3.2) Võimendus jäetakse 1_5, kuna katseeksituse meetodil see sobis tulemusega.

```
const struct device *adc_dev;

static const struct adc_channel_cfg m_1st_channel_cfg = {
    .gain = ADC_GAIN,
    .reference = ADC_REFERENCE,
    .acquisition_time = ADC_ACQUISITION_TIME,
    .channel_id = ADC_CHANNEL_ID,
#if defined(CONFIG_ADC_CONFIGURABLE_INPUTS)
    .input_positive = ADC_CHANNEL_INPUT,
#endif

};
```

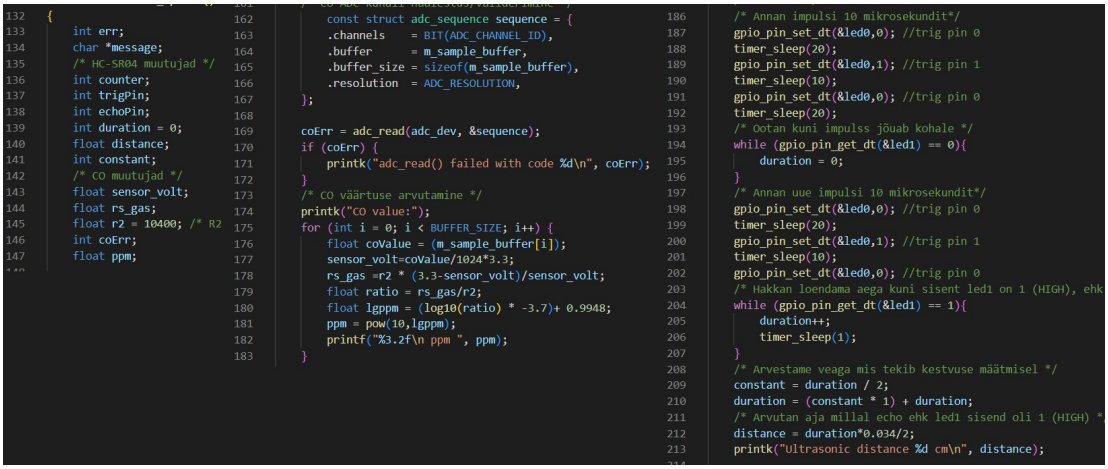
```
//CO
#define ADC_DEVICE_NODE     DT_INST(
#define ADC_RESOLUTION      10
#define ADC_GAIN        ADC_GAIN_1_5
#define ADC_REFERENCE       ADC_REF_I
#define ADC_ACQUISITION_TIME    ADC_A
#define ADC_CHANNEL_ID  0
#define ADC_CHANNEL_INPUT   NRF_SAADC
#define BUFFER_SIZE 1
//CO
const struct device *adc_dev;
```

Go to Definition
Go to Declaration
Go to Type Definition
Go to References
Peek
Find All References
Rename Symbol
Change All Occurrences

Joonis 7.3 ADC seadistused [32]

Kui mingi seadistuse tähendust ei teadnud ja oli vaja seda uurida lähemalt või vaadata mis võimalusi oli veel võimalik kasutada siis sai hiirega minna soovitud valiku peale, teha parem hiire klik ja valida *Go to Defination*. Antud tegevus viis .h laiendusega faili kus oli seletused koos võimalike teiste variantidega olemas.

Koodi põhiosa sai lisatud *static int sharow_update()* alla mille algusesse lisati muutujad (vt. Joonis 7.4) ja koodi põhiosa mida sai veidike muudetud individuaalsetest koodidest.



```
{
    int err;
    char *message;
    /* HC-SR04 muutujad */
    int counter;
    int trigPin;
    int echoPin;
    int duration = 0;
    float distance;
    int constant;
    /* CO muutujad */
    float sensor_volt;
    float rs_gas;
    float r2 = 10400; /* R2
    int coErr;
    float ppm;
```

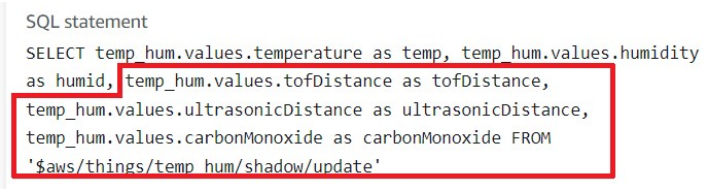
```
        const struct adc_sequence sequence = {
        .channels    = BIT(ADC_CHANNEL_ID),
        .buffer      = m_sample_buffer,
        .buffer_size = sizeof(m_sample_buffer),
        .resolution  = ADC_RESOLUTION,
    };

    coErr = adc_read(adc_dev, &sequence);
    if (coErr) {
        printk("adc_read() failed with code %d\n", coErr);
    }
    /* CO väärtuse arvutamine */
    printk("CO value:");
    for (int i = 0; i < BUFFER_SIZE; i++) {
        float coValue = (m_sample_buffer[i]);
        sensor_volt=coValue/1024*3.3;
        rs_gas =r2 * (3.3-sensor_volt)/sensor_volt;
        float ratio = rs_gas/r2;
        float lgppm = (log10(ratio) * -3.7)+ 0.9948;
        ppm = pow(10,lgppm);
        printf("%3.2f\n ppm ", ppm);
    }
```

```
    /* Annan impulsi 10 mikrosekundit*/
    gpio_pin_set_dt(&led0,0); //trig pin 0
    timer_sleep(20);
    gpio_pin_set_dt(&led0,1); //trig pin 1
    timer_sleep(10);
    gpio_pin_set_dt(&led0,0); //trig pin 0
    timer_sleep(20);
    /* Ootan kuni impulss jõuab kohale */
    while (gpio_pin_get_dt(&led1) == 0){
        duration = 0;
    }
    /* Annan uue impulsi 10 mikrosekundit*/
    gpio_pin_set_dt(&led0,0); //trig pin 0
    timer_sleep(20);
    gpio_pin_set_dt(&led0,1); //trig pin 1
    timer_sleep(10);
    gpio_pin_set_dt(&led0,0); //trig pin 0
    /* Hakkan loendama aega kuni sisent led1 on 1 (HIGH), ehk
    while (gpio_pin_get_dt(&led1) == 1){
        duration++;
        timer_sleep(1);
    }
    /* Arvestame veaga mis tekib kestvuse määtmisel */
    constant = duration / 2;
    duration = (constant * 1) + duration;
    /* Arvutan aja millal echo ehk led1 sisend oli 1 (HIGH) *
    distance = duration*0.034/2;
    printk("Ultrasonic distance %d cm\n", distance);
```

Joonis 7.4 koodi arvutuslik osa CO ja HC-SR04 [33]

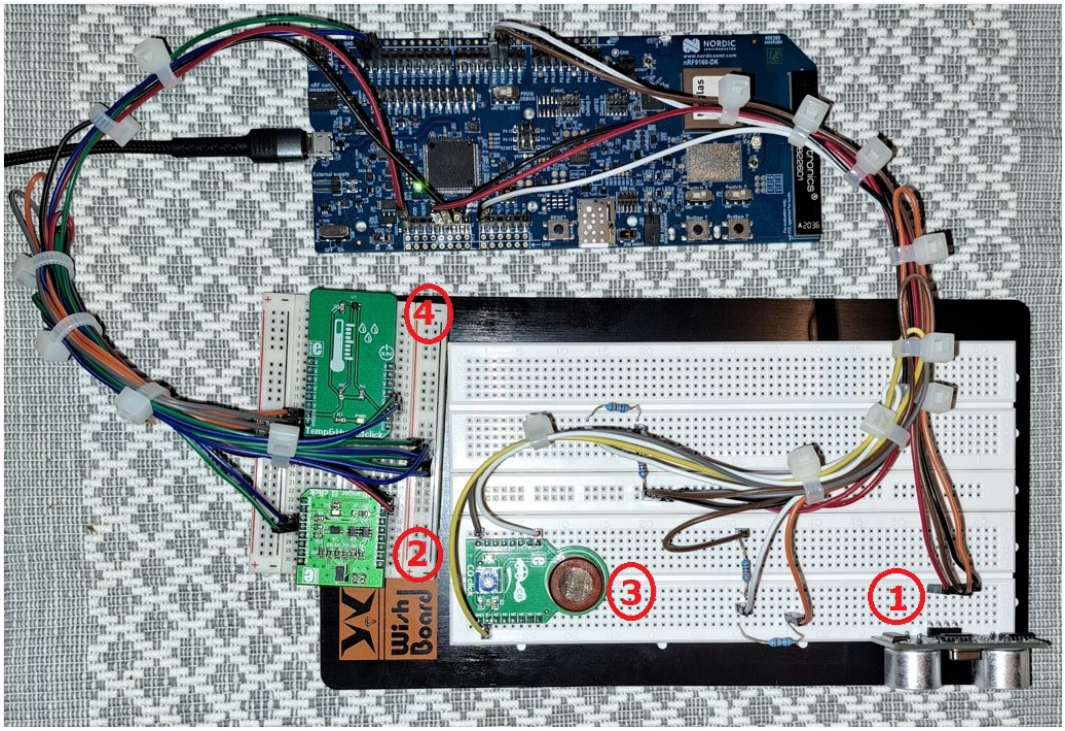
Muudatused tehti ka *void main (void)* koodi osas kus määratakse sisendid väljundid ja kanalid seadistatakse, ning kui midagi ei tööta edastatakse viga (vt. Lisa 6). Kõik uute andurite andmed lisati JSON formaati ja edastati AWS IoT-le.



```
SQL statement
SELECT temp_hum.values.temperature as temp, temp_hum.values.humidity
as humid, temp_hum.values.tofDistance as tofDistance,
temp_hum.values.ultrasonicDistance as ultrasonicDistance,
temp_hum.values.carbonMonoxide as carbonMonoxide FROM
'$aws/things/temp_hum/shadow/update'
```

Joonis 7.5 reegel andmete edastamiseks andmebaasi

AWS IoT-s tuli märkida eelnevalt loodud teema *$aws/things/temp_hum/shadow/update* sõnumite edastamiseks andmebaasi reeglile juurde ka uute andurite andmete edastamine. Selleks oli vaja lisada *SQL* päringusse mõned päringud juurde (vt. Joonis 7.5).



Joonis 7.6 Andurite paigutus maketeerimisplaadil

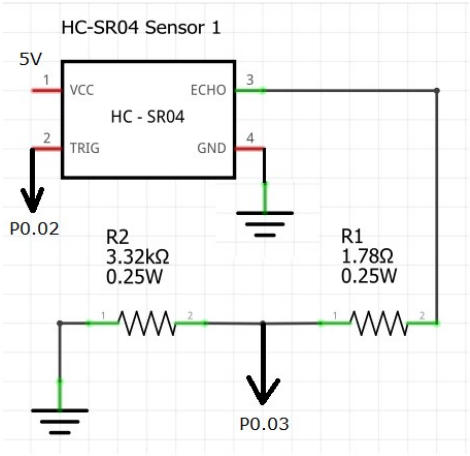
Jooniselt (vt. Joonis 7.6) on näha andurite paiguts maketerimislaual:

1. Ultraheliandur HC-SR04;
2. ToF VL53L0X andur;
3. Vingugaasiandur MQ-7;
4. Temperatuuri-ja niiskuseandur ens210.

7.1 Ultraheliandurи HC-SR04 integreerimine digitaalse sisendi ja väljundiga

Ultraheliandurи integreerimiseks arendusplaadiga tuli selgeks teha tööpõhimõte, milleks oli siis helilevimisekiirus tubastes tingimustes. Teades eelnevatest füüsika tundidest, et helilevimise kiirus on ligikaudu 340 meetrit sekundis siis sellest lähtuvalt saab arvutada kauguse objektist kasutades valemit helilevimise kiirus korda aeg mis selleks kulub. Kuna helilained levivad edasi ja tagasi, ehk aeg on kahekordne kui jõuavad vastuvõtjasse siis tuleb ka tulemus kahega jagada. Andur sai paugutatud maketeerimislaua nurka kus teised detailid ei seganud anduri tööd (vt Joonis 7.6)

Kuna andur töötas 5V toitepingega ja arendusplaadi sisendisse ei tohi lasta sellist pinget sai loodud pingejagur maketeerimislaual mis hoidis ära, et pinge üle lubatud väärtuste ei tõuseks milleks on 3.3V. Selleks sai kasutatud kahte takistit mis on 1% tolerantsiga R2 3.32 k ja R1 1.78 k (vt Joonis 7.7).



Joonis 7.7 HC-SR04 ühendusskeem

Sisend P0.03 on digitaalne sisend ehk ta rakendab koodis loenduri kui tuleb pinge peale sisendile. Samuti antud sisendi rakendumine on nähtav arendusplaadi LED2 vaadates, LED2 põleb seni kui signaal on peal (vt. Joonis 3.2). Signaali genereerib väljund P0.02 millest annab ka märku LED1 (vt. Joonis 3.2) Signaali 3.3V pinege antakse impulsina 10 mikrosekundit mida reguleeritakse taimeriga. Anduri kood töötamaks arendusplaadiga on leitav (vt. Lisa 3).

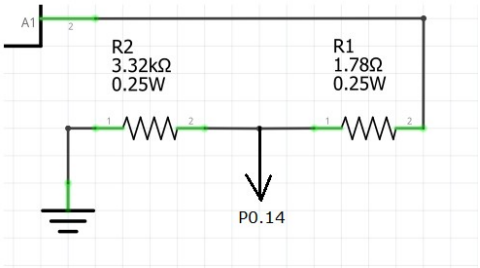
7.2 ToF anduri VL53L0X integreerimine I2C liiniga

Andur suhtleb üle *I2C* liini seega tema lisamine käib sarnaselt nagu temperatuuri-ja niiskuseanduri ens210 lisamine, lihtsalt aardess on teine, oli leitav dokumentatsioonist 0x29. Toide 3.3V võetakse paraleelselt sealtki kus saab toite ens210 andur ja *SDA* ning *SCL* ühendatakse väljaviikude P0.30 ja P031 külge (vt. Joonis 6.5).

Arendusplaadi SDK draiverite all oli kood millega oli võimalik lugeda andmeid andurilt vastavate päringute saatmisega koodis (vt. Lisa 4), millede väärtusi kombineerides *val1* ja *val2* oli võimalik kätte saada näit meetrites.

7.3 Vingugaasianduri MQ-7 integreerimine analoogsisendiga

Sellel anduril on kasutuses analoog väljund, mis tähendab, et arendusplaadil peame kasutama analoogsisendit. Vastavalt anduri spetsifikatsioonile kasutab toitepinget 5V ja väljund annab samamoodi kuni 5V välja. Sarnaselt ultraheliandurile tuleb ka siin kasutada pingejagurit, et ühendada arendusplaadi sisendiga P0.14 (vt. Joonis 3.2 ja Joonis 7.8).



Joonis 7.8 Vingugaasianduri ühendamine arendusplaadi sisendiga

Kuna andur tuvastab vahemikus 10-500 ppm siis piisab 10 bitisest sisendist ehk meil on 3.3V juures 1024 erinevat näitu. Kuna protsessor ei oska analogsignaaliga midagi teha siis muundatakse signaal digitaalseks analog/digitaal konverteriga mis on arendusplaadil sisseehitatud. Mis tähendab seda, et iga ligikaudu 0.0032 V muutus muudab meie näitu 1 võrra suuremaks või väiksemaks. Näiteks kui meil on näit analog/digitaal konverterist mis tuleb välja 400 siis korrutame selle 0.0032 ja saame teada mis on pige meie sisendis P0.14 ehk 1.28V antud näitega. Saadud pingest on siis võimalik tuletada väärtus mida andur peaks näitama.

8 TULEMUSED

Antud lõputöö käigus toimus pidev enesearendamine ja õppimine, millega kaasnesid ka erinevad probleemid. Ultraheliandurі täpsusklass jäi meetri ulatuses 3 mm juurde, aga 2.7 m juures oli viga umbes 2 cm. Viga jäi sisse sellepärast kuna ei jõudnud luua sellist taimerit mis oleks olnud suuteline lugema 1 mikrosekundilise täpsusega impulsi aega sisendis mille külge oli ühendatud *echo* jalg ultraheliandurіl.

Vingugaasi anduri kontrollimiseks kasutasin purki ja küünalt, et tekitada hapnikuvaeses keskkonnas põlemisel vingugaasi. Kui küünal põles asetasin purgi küünla peale. Peale küünla kustumist asetasin purgi andurile peale ja näit hakkas tõusma, mõne ühiku võrra. Sellega lugesin katsetuse läbinuks ja andur tundus töötavat.

Temperatuuri-ja niiskuseanduri katsetamine toimus toatemperatuuril siseruumides. Näite võrdlesin omavahel ilmajaama näitudega, mis kinnitasid anduri korrektset mõõtetulemust. Temperatuuri tõusu ja langust sai proovitud kui andur katta näpuga kinni mille järel temperatuuri näit hakkas tõusma. Õhuniiskuse muutmiseks tuli lihtsalt hingata anduri peale, peale mida hakkas näit suurenema .

ToF anduri katsetamine toimus objektide asetamisega mõõdulindil erinevate kaugusteni, kus mõõtetäpsus oli 1 mm lauapeal mõõtes kuni 1.7 meetrini. Kaugemal mõjutas mõõtetulemust peegeldava pinna värv ja toa valgustus.

Arendusplaadi ühendus võrguga oli stabiilne ja katsetuste käigus ei katkenud. Maksimaalne aeg mis prooviti oli 1 päev ja edastati andmeid minutilise intervalliga. Samamoodi töötasid kõik kasutatud AWS teenused ilusti ja andmed visualiseeriti Grafanaga vastavalt saadetud andmetele arendusplaadi poolt (vt. Lisa 7).

9 PROJEKTI EDASIARENDAMISE VÕIMALUSED

Kuna MQTT osa antud diplomitöös on kohandatud spetsiaalselt AWS keskonda silmas pidades milles andmemahu kasvades teenused muutuvad tasuliseks ja tasu suurus sõltub mahtudest. Teatud edasiarenduste käigus võivad rakenduda tasud, näiteks kui on soov saada AWS IoT keskkonnast andmeid kolmandatele osapooltele.

Antud projektile on võimalik juurde arendada veebirakendus, kuna andmeid on võimalik saata ka HTTPS POST meetodiga AWS IoT Core keskkonnast enda loodud veebirakendusele, mis küll on tasuline teenus ja hind sõltub andmemahtudest. Veebirakendus oleks tavakasutajale tunduvalt lihtsam viis andmete lugemiseks mida arendusplaat edastab ja osavalt kujundatud rakendus lihtsustaks andmete lugemist.

Teise võimalusena antud tööd edasi arendada on võimalik luua ise MQTT maakler, milleks siis vajalik serveri seadistamine kus MQTT maakler hakkab tööle. Tasuta variantidest on näiteks Eclipse Mosquitto, millega muidugi tuleb MQTT ühenduse koodiosa ümber kirjutada. Millega on võimalik saata siis andmed edasi näiteks Postgre andmebaasi. Antud juhul tuleks MQTT koodiosa ümber teha teise keskkonna jaoks, näiteks on võimalik võtta SDK-s pakutav MQTT_sample mall.

AWS-is on võimalik kogutud andmeid töödelda ja nende põhjal teha andmeanalüüsi kasutades keskkonda AWS IoT Analytics mille tulemusena saadakse andmestik ja sarnaselt R-Studiole on võimalik teha andmeanalüüsi nende andmetega saades samamoodi erinevaid graafikuid või teisi vajalike näitajaid kätte, selleks oleks muidugi vaja suuremaid andmete hulkasid, et tulemus jääks ilusam, samuti võivad lisanduda lisa tasud.

KOKKUVÕTE

Põhieesmärgid said lõputöös täidetud, andmete edastamiseks ühendati erinevad andurid juhtmeid ja maketeerimisaluseid kasutades arendusplaadiga. Kõik andurid said toite arendusplaadilt mis omakorda sai toite üle USB kaabli arvutist. Andurid ühendati kasutades erinevaid sisendeid ja väljundeid arendusplaadil.

Andmete edastamiseks kasutati raadiosidevõrku NB-IoT mis on litsentseeritud ja turvaline kuna andmeedastus on krüpteeritud. Võrguühenduse loomiseks teenusepakkuja võrguga pidi soetama SIM kaadi mis toetas NB-IoT võrku.

Andurite näitude lugemiseks katsetati igat andurit eraldi, milleks loodi eraldi kood arendusplaadile, mis hiljem lisati põhikoodi vastavatesse osadesse juurde mis edastas andmeid AWS IoT Core-le. Andmed edastati MQTT protokolli kasutades, kuna on väikese koodijalajäljega ja sobib energiasäästlikuks andmeedastuseks.

AWS keskkonnas loodi objekt millele hakati saatma JSON formaadis andmeid konkreetse teema alla *$aws/things/temp_hum/shadow/Update.* Selleks loodi eelnevalt poliitika mis lubab IoT seadmel saata andmeid. Hiljem juba sai saadud andmeid AWS erinevate keskkondade vahel edastada luues vastavaid reegleid. Andmete visualiseerimine toimus Grafana abiga milles olid erinevate andurite näidud graafikute ja näidikuna.

Tulemusena valmis arendusplaadi ja andurite vahelised füüsilised ühendused maketeerimisalusel. Loodi programm mis edastas iga minut andmeid AWS keskkonda andurite hetke väärtustest, mida töödeldi edasi erinevate AWS keskkondade vahel kuni saadi andmetest graafikud ajas.

Töö käigus õpiti kasutama võtteid mobiiliseadmete programmeerimiseks, arendusplaati programmeeriti C keeles. Suheldi foorumi vahendusel Nordic Semiconductori tehnilise toega saamaks tehnilist nõu seadme kohta, konsulteeriti töö tulemuste osas juhendajaga. Saadud uued teadmised olid suuresti programmeerimiskeeles C ja kuidas rakendada C keelt mobiiliseadmete programmeerimiseks.

Antud lõputööd on võimalik edasi arendada ja kasutada mõne targa linna lahenduses, näiteks prügikonteinerite mahtuvuse mõõtmiseks, et teada saada millal konteinerid vajavad tühjendamist. Samuti võib lisada arendusplaadile ka teisi andureid või olemasolevad välja vahetada ja kasutusala vastavalt sellele muuta.

SUMMARY

This thesis is relevant because the use of IoT devices in today's world is almost in every sphere of life, and the share of IoT devices is constantly growing and also needs energy-saving solutions. The topic of the thesis is Using NB-IoT radio communication to measure environmental parameters, authored by Roald Põllu.

The NB-IoT network helps to create a secure and fast communication between the device and the end user, which in turn makes it possible to create different smart city solutions using different sensors. For example, parking, waste management, street lighting and other smart city applications.

Main tasks:

- Examining the hardware of the development board and installing the necessary programs;
- Integration of sensors with various inputs and outputs of the development board;
- Data transmission using radio communication;
- Data storage in the database and visualization.

To obtain data, various sensors are used, which are connected to the inputs and outputs of the development board with cables on the mock-up base, and the received data is transmitted at minute intervals to the cloud platform, where the data is sent to the database and the data is taken from there for visualization. The result is data that changes over time, which is visualized as graphs and indicators.

To read the readings of the sensors, each sensor was tested separately, for which a separate code was created on the development board, which was later added to the corresponding parts of the main code that transmitted the data to the AWS IoT Core. The data was transmitted using the MQTT protocol, as it has a small code footprint and is suitable for energy-efficient data transmission.

In the AWS environment, an object was created to which data in JSON format was sent under a specific topic $aws/things/temp_hum/shadow/Update. For this purpose, a policy was created in advance that allows the IoT device to send data. Later, the obtained data could be transferred between different AWS environments by creating the corresponding rules. Data visualization was done with the help of Grafana, which had the readings of various sensors as graphs and indicators.

As a result, the physical connections between the development board and the sensors on the mock-up base are ready. A program was created that transmitted data from

the current values of the sensors to the AWS environment every minute, which was further processed between different AWS environments until graphs over time were obtained from the data.

KASUTATUD KIRJANDUSE LOETELU

1. Nordic Semiconductor, Glossary [Online] https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/glossary.html (12.09.2022)

2. Paessler AG, IT Explained: MQTT [*Online*] https://www.paessler.com/it-explained/mqtt (03.10.2022)

3. T-Mobile USA, What is LTE [*Online*] https://www.t-mobile.com/resources/what-is-lte (17.09.2022)

4. Nordic Semiconductor, nRF9160 DK Hardware v1.1.0 User Guide [*Online*] https://infocenter.nordicsemi.com/pdf/nRF9160_DK_HW_User_Guide_v1.1.0.pdf (13.09.2022)

5. ZIFF DAVIS. PCMAG DIGITAL GROUP, GSM [*Online*] https://www.pcmag.com/encyclopedia/term/gsm (05.10.2022)

6. Achiv Chauhan, Palak Jain, TCP/IP Model [*Online*] https://www.geeksforgeeks.org/tcp-ip-model/ (05.10.2022)

7. Arpit Asati, What is web socket and how it is different from the HTTP? [*Online*] https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/?ref=gcse (06.10.2022)

8. ZIFF DAVIS. PCMAG DIGITAL GROUP, HTTPS [*Online*] https://www.pcmag.com/encyclopedia/term/https (06.10.2022)

9. Roald Põllu, Dejans, Visual Studio Code and library libpq [*Online*] https://devzone.nordicsemi.com/f/nordic-q-a/92280/visual-studio-code-and-library-libpq (26.09.2022)

10. Alexander S. Gillis, What is JSON (JavaScript Object Notation)? [*Online*] https://www.theserverside.com/definition/JSON-Javascript-Object-Notation (18.10.2022)

11. Computer Hope, USB [*Online*] https://www.computerhope.com/jargon/u/usb.htm (18.10.2022)

12. Peter Loshin, What is Structured Query Language (SQL)? [*Online*] https://www.techtarget.com/searchdatamanagement/definition/SQL

13. STMicroelectronics, VL53L0X [*Online*] https://download.mikroe.com/documents/datasheets/vl53l0x-sensor-datasheet.pdf (07.11.2022)

14. Circuit Basics, BASICS OF THE I2C COMMUNICATION PROTOCOL [*Online*] https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ (26.10.2022)

15. TechGenix, A Guide to NACK/NAK (Negative Acknowledgment or Not Acknowledged) [*Online*] https://techgenix.com/nack-nak-negative-acknowledgement-guide/ (07.11.22)

16. Toby Jaffey, MQTT and CoAP, IoT Protocols [*Online*] https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php (14.11.2022)

17. AGUSTIN PELAEZ, LoRaWAN vs NB-IoT: A Comparison Between IoT Trend-Setters [*Online*] https://ubidots.com/blog/lorawan-vs-nb-iot/ (14.11.2022)

18. Gus Vos, What is Narrowband IoT (NB-IoT)? [*Online*] https://www.sierrawireless.com/iot-blog/what-is-nb-iot/ (17.09.2022)

19. Jessica Hopkins, Understanding the Differences Between UART and I2C https://www.totalphase.com/blog/2020/12/differences-between-uart-i2c/ (14.11.2022)

20. Arduino, MKR NB 1500 https://docs.arduino.cc/hardware/mkr-nb-1500 [*Online*] (10.12.2022)

21. Nordic Semiconductor, nRF9160 Product Specification https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf91_dk%2FUG%2Fnrf91_DK%2Fintro.html [Online] (10.12.2022)

22. Nordic Semiconductor, nRF9160 DK Development kit [*Online*] https://www.nordicsemi.com/Products/Development-hardware/nRF9160-DK/Download?lang=en#infotabs (12.09.2022)

23. Nordic Semiconductor, nRF Connect for Desktop Apps [*Online*] https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-desktop (13.09.2022)

24. Nordic Semiconductor, nRF91 AT Commands, Command Reference Guide v2.1 [*Online*] https://infocenter.nordicsemi.com/pdf/nrf91_at_commands_v2.1.pdf (19.09.2022)

25. Nordic Semiconductor, Network mode [*Online*] https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.1.0/nrf/ug_nrf91_features.html#nrf9160-ug-band-lock (15.09.2022)

26. MikroElektronika, Temp&Hum 6 Click datasheets [*Online*] https://www.farnell.com/datasheets/2773931.pdf (05.10.2022)

27. Elecfreaks, Ultrasonic Ranging Module HC - SR04 [*Online*] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf (03.11.2022)

28. MikroElektronika, CO CLICK [*Online*] https://www.mikroe.com/co-click (03.11.2022)

29. GreeksforGreeks, Comparisons between Azure Vs AWS https://www.geeksforgeeks.org/comparisons-between-azure-vs-aws/ [*Online*] (08.12.2022)

30. Amazon Web Services, What is AWS IoT? [*Online*] https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (28.09.2022)

31. Zephyr® Project, a Linux Foundation Project, About the Zephyr Project [*Online*] https://www.zephyrproject.org/learn-about/ (20.09.2022)

32. Kenneth , nrf9160 DK Analog Input Pin Configuration https://devzone.nordicsemi.com/f/nordic-q-a/92914/nrf9160-dk-analog-input-pin-configuration [*Online*] (27.10.2022)

33. betuse MQ7 gas sensor TEMP and HUM compensation https://forum.arduino.cc/t/mq7-gas-sensor-temp-and-hum-compensation/416439/3 [*Online*] (07.11.2022)

34. Håkon Alseth, Create accurate microsecond timer without interrupts https://devzone.nordicsemi.com/f/nordic-q-a/58621/create-accurate-microsecond-timer-without-interrupts/238258 [*Online*] (10.11.2022)

Lisa 1 - AWS_iot mall SDK vrsioonist 2.1.0

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-Nordic-5-Clause
 */

#include <zephyr/kernel.h>
#include <stdio.h>
#include <stdlib.h>
#if defined(CONFIG_NRF_MODEM_LIB)
#include <modem/lte_lc.h>
#include <modem/nrf_modem_lib.h>
#include <modem/modem_info.h>
#include <nrf_modem.h>
#endif
#include <net/aws_iot.h>
#include <zephyr/sys/reboot.h>
#include <date_time.h>
#include <zephyr/dfu/mcuboot.h>
#include <cJSON.h>
#include <cJSON_os.h>

BUILD_ASSERT(!IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT),
        "This sample does not support LTE auto-init and connect");

#define APP_TOPICS_COUNT CONFIG_AWS_IOT_APP_SUBSCRIPTION_LIST_COUNT

static struct k_work_delayable shadow_update_work;
static struct k_work_delayable connect_work;
static struct k_work shadow_update_version_work;

static bool cloud_connected;

static K_SEM_DEFINE(lte_connected, 0, 1);
static K_SEM_DEFINE(date_time_obtained, 0, 1);

#if defined(CONFIG_NRF_MODEM_LIB)
NRF_MODEM_LIB_ON_INIT(aws_iot_init_hook, on_modem_lib_init, NULL);

/* Initialized to value different than success (0) */
static int modem_lib_init_result = -1;

static void on_modem_lib_init(int ret, void *ctx)
{
    modem_lib_init_result = ret;
}
```

```
#endif

static int json_add_obj(cJSON *parent, const char *str, cJSON *item)
{
    cJSON_AddItemToObject(parent, str, item);

    return 0;
}

static int json_add_str(cJSON *parent, const char *str, const char *item)
{
    cJSON *json_str;

    json_str = cJSON_CreateString(item);
    if (json_str == NULL) {
        return -ENOMEM;
    }

    return json_add_obj(parent, str, json_str);
}

static int json_add_number(cJSON *parent, const char *str, double item)
{
    cJSON *json_num;

    json_num = cJSON_CreateNumber(item);
    if (json_num == NULL) {
        return -ENOMEM;
    }

    return json_add_obj(parent, str, json_num);
}

static int shadow_update(bool version_number_include)
{
    int err;
    char *message;
    int64_t message_ts = 0;
    int16_t bat_voltage = 0;

    err = date_time_now(&message_ts);
    if (err) {
        printk("date_time_now, error: %d\n", err);
        return err;
    }

#if defined(CONFIG_NRF_MODEM_LIB)
    /* Request battery voltage data from the modem. */
    err = modem_info_short_get(MODEM_INFO_BATTERY, &bat_voltage);
```

```
    if (err != sizeof(bat_voltage)) {
        printk("modem_info_short_get, error: %d\n", err);
        return err;
    }
#endif

    cJSON *root_obj = cJSON_CreateObject();
    cJSON *state_obj = cJSON_CreateObject();
    cJSON *reported_obj = cJSON_CreateObject();

    if (root_obj == NULL || state_obj == NULL || reported_obj == NULL) {
        cJSON_Delete(root_obj);
        cJSON_Delete(state_obj);
        cJSON_Delete(reported_obj);
        err = -ENOMEM;
        return err;
    }

    if (version_number_include) {
        err = json_add_str(reported_obj, "app_version",
                    CONFIG_APP_VERSION);
    } else {
        err = 0;
    }

    err += json_add_number(reported_obj, "batv", bat_voltage);
    err += json_add_number(reported_obj, "ts", message_ts);
    err += json_add_obj(state_obj, "reported", reported_obj);
    err += json_add_obj(root_obj, "state", state_obj);

    if (err) {
        printk("json_add, error: %d\n", err);
        goto cleanup;
    }

    message = cJSON_Print(root_obj);
    if (message == NULL) {
        printk("cJSON_Print, error: returned NULL\n");
        err = -ENOMEM;
        goto cleanup;
    }

    struct aws_iot_data tx_data = {
        .qos = MQTT_QOS_0_AT_MOST_ONCE,
        .topic.type = AWS_IOT_SHADOW_TOPIC_UPDATE,
        .ptr = message,
        .len = strlen(message)
    };
```

```
    printk("Publishing: %s to AWS IoT broker\n", message);

    err = aws_iot_send(&tx_data);
    if (err) {
        printk("aws_iot_send, error: %d\n", err);
    }

    cJSON_FreeString(message);

cleanup:

    cJSON_Delete(root_obj);

    return err;
}

static void connect_work_fn(struct k_work *work)
{
    int err;

    if (cloud_connected) {
        return;
    }

    err = aws_iot_connect(NULL);
    if (err) {
        printk("aws_iot_connect, error: %d\n", err);
    }

    printk("Next connection retry in %d seconds\n",
           CONFIG_CONNECTION_RETRY_TIMEOUT_SECONDS);

    k_work_schedule(&connect_work,
            K_SECONDS(CONFIG_CONNECTION_RETRY_TIMEOUT_SECONDS));
}

static void shadow_update_work_fn(struct k_work *work)
{
    int err;

    if (!cloud_connected) {
        return;
    }

    err = shadow_update(false);
    if (err) {
        printk("shadow_update, error: %d\n", err);
    }
```

```
    printk("Next data publication in %d seconds\n",
           CONFIG_PUBLICATION_INTERVAL_SECONDS);

    k_work_schedule(&shadow_update_work,
            K_SECONDS(CONFIG_PUBLICATION_INTERVAL_SECONDS));
}

static void shadow_update_version_work_fn(struct k_work *work)
{
    int err;

    err = shadow_update(true);
    if (err) {
        printk("shadow_update, error: %d\n", err);
    }
}

static void print_received_data(const char *buf, const char *topic,
                size_t topic_len)
{
    char *str = NULL;
    cJSON *root_obj = NULL;

    root_obj = cJSON_Parse(buf);
    if (root_obj == NULL) {
        printk("cJSON Parse failure");
        return;
    }

    str = cJSON_Print(root_obj);
    if (str == NULL) {
        printk("Failed to print JSON object");
        goto clean_exit;
    }

    printf("Data received from AWS IoT console:\nTopic: %.*s\nMessage: %s\n",
           topic_len, topic, str);

    cJSON_FreeString(str);

clean_exit:
    cJSON_Delete(root_obj);
}

void aws_iot_event_handler(const struct aws_iot_evt *const evt)
{
    switch (evt->type) {
    case AWS_IOT_EVT_CONNECTING:
        printk("AWS_IOT_EVT_CONNECTING\n");
```

```
        break;
    case AWS_IOT_EVT_CONNECTED:
        printk("AWS_IOT_EVT_CONNECTED\n");

        cloud_connected = true;
        /* This may fail if the work item is already being processed,
         * but in such case, the next time the work handler is executed,
         * it will exit after checking the above flag and the work will
         * not be scheduled again.
         */
        (void)k_work_cancel_delayable(&connect_work);

        if (evt->data.persistent_session) {
            printk("Persistent session enabled\n");
        }

#if defined(CONFIG_NRF_MODEM_LIB)
        /** Successfully connected to AWS IoT broker, mark image as
         *  working to avoid reverting to the former image upon reboot.
         */
        boot_write_img_confirmed();
#endif

        /** Send version number to AWS IoT broker to verify that the
         *  FOTA update worked.
         */
        k_work_submit(&shadow_update_version_work);

        /** Start sequential shadow data updates.
         */
        k_work_schedule(&shadow_update_work,
                K_SECONDS(CONFIG_PUBLICATION_INTERVAL_SECONDS));

#if defined(CONFIG_NRF_MODEM_LIB)
        int err = lte_lc_psm_req(true);
        if (err) {
            printk("Requesting PSM failed, error: %d\n", err);
        }
#endif
        break;
    case AWS_IOT_EVT_READY:
        printk("AWS_IOT_EVT_READY\n");
        break;
    case AWS_IOT_EVT_DISCONNECTED:
        printk("AWS_IOT_EVT_DISCONNECTED\n");
        cloud_connected = false;
        /* This may fail if the work item is already being processed,
         * but in such case, the next time the work handler is executed,
         * it will exit after checking the above flag and the work will
```

```
         * not be scheduled again.
         */
        (void)k_work_cancel_delayable(&shadow_update_work);
        k_work_schedule(&connect_work, K_NO_WAIT);
        break;
    case AWS_IOT_EVT_DATA_RECEIVED:
        printk("AWS_IOT_EVT_DATA_RECEIVED\n");
        print_received_data(evt->data.msg.ptr, evt->data.msg.topic.str,
                    evt->data.msg.topic.len);
        break;
    case AWS_IOT_EVT_PUBACK:
        printk("AWS_IOT_EVT_PUBACK, message ID: %d\n", evt->data.message_id);
        break;
    case AWS_IOT_EVT_FOTA_START:
        printk("AWS_IOT_EVT_FOTA_START\n");
        break;
    case AWS_IOT_EVT_FOTA_ERASE_PENDING:
        printk("AWS_IOT_EVT_FOTA_ERASE_PENDING\n");
        printk("Disconnect LTE link or reboot\n");
#if defined(CONFIG_NRF_MODEM_LIB)
        err = lte_lc_offline();
        if (err) {
            printk("Error disconnecting from LTE\n");
        }
#endif
        break;
    case AWS_IOT_EVT_FOTA_ERASE_DONE:
        printk("AWS_FOTA_EVT_ERASE_DONE\n");
        printk("Reconnecting the LTE link");
#if defined(CONFIG_NRF_MODEM_LIB)
        err = lte_lc_connect();
        if (err) {
            printk("Error connecting to LTE\n");
        }
#endif
        break;
    case AWS_IOT_EVT_FOTA_DONE:
        printk("AWS_IOT_EVT_FOTA_DONE\n");
        printk("FOTA done, rebooting device\n");
        aws_iot_disconnect();
        sys_reboot(0);
        break;
    case AWS_IOT_EVT_FOTA_DL_PROGRESS:
        printk("AWS_IOT_EVT_FOTA_DL_PROGRESS, (%d%%)",
               evt->data.fota_progress);
    case AWS_IOT_EVT_ERROR:
        printk("AWS_IOT_EVT_ERROR, %d\n", evt->data.err);
        break;
    case AWS_IOT_EVT_FOTA_ERROR:
```

```
        printk("AWS_IOT_EVT_FOTA_ERROR");
        break;
    default:
        printk("Unknown AWS IoT event type: %d\n", evt->type);
        break;
    }
}

static void work_init(void)
{
    k_work_init_delayable(&shadow_update_work, shadow_update_work_fn);
    k_work_init_delayable(&connect_work, connect_work_fn);
    k_work_init(&shadow_update_version_work, shadow_update_version_work_fn);
}

#if defined(CONFIG_NRF_MODEM_LIB)
static void lte_handler(const struct lte_lc_evt *const evt)
{
    switch (evt->type) {
    case LTE_LC_EVT_NW_REG_STATUS:
        if ((evt->nw_reg_status != LTE_LC_NW_REG_REGISTERED_HOME) &&
             (evt->nw_reg_status != LTE_LC_NW_REG_REGISTERED_ROAMING)) {
            break;
        }

        printk("Network registration status: %s\n",
            evt->nw_reg_status == LTE_LC_NW_REG_REGISTERED_HOME ?
            "Connected - home network" : "Connected - roaming");

        k_sem_give(&lte_connected);
        break;
    case LTE_LC_EVT_PSM_UPDATE:
        printk("PSM parameter update: TAU: %d, Active time: %d\n",
            evt->psm_cfg.tau, evt->psm_cfg.active_time);
        break;
    case LTE_LC_EVT_EDRX_UPDATE: {
        char log_buf[60];
        ssize_t len;

        len = snprintf(log_buf, sizeof(log_buf),
                   "eDRX parameter update: eDRX: %f, PTW: %f",
                   evt->edrx_cfg.edrx, evt->edrx_cfg.ptw);
        if (len > 0) {
            printk("%s\n", log_buf);
        }
        break;
    }
    case LTE_LC_EVT_RRC_UPDATE:
        printk("RRC mode: %s\n",
```

```
            evt->rrc_mode == LTE_LC_RRC_MODE_CONNECTED ?
            "Connected" : "Idle");
        break;
    case LTE_LC_EVT_CELL_UPDATE:
        printk("LTE cell changed: Cell ID: %d, Tracking area: %d\n",
            evt->cell.id, evt->cell.tac);
        break;
    default:
        break;
    }
}

static void modem_configure(void)
{
    int err;

    if (IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT)) {
        /* Do nothing, modem is already configured and LTE connected. */
    } else {
        err = lte_lc_init_and_connect_async(lte_handler);
        if (err) {
            printk("Modem could not be configured, error: %d\n",
                err);
            return;
        }
    }
}

static void nrf_modem_lib_dfu_handler(void)
{
    int err;

    err = modem_lib_init_result;

    switch (err) {
    case MODEM_DFU_RESULT_OK:
        printk("Modem update suceeded, reboot\n");
        sys_reboot(SYS_REBOOT_COLD);
        break;
    case MODEM_DFU_RESULT_UUID_ERROR:
    case MODEM_DFU_RESULT_AUTH_ERROR:
        printk("Modem update failed, error: %d\n", err);
        printk("Modem will use old firmware\n");
        sys_reboot(SYS_REBOOT_COLD);
        break;
    case MODEM_DFU_RESULT_HARDWARE_ERROR:
    case MODEM_DFU_RESULT_INTERNAL_ERROR:
        printk("Modem update malfunction, error: %d, reboot\n", err);
        sys_reboot(SYS_REBOOT_COLD);
```

```
        break;
    default:
        break;
    }
}
#endif

static int app_topics_subscribe(void)
{
    int err;
    static char custom_topic[75] = "my-custom-topic/example";
    static char custom_topic_2[75] = "my-custom-topic/example_2";

    const struct aws_iot_topic_data topics_list[APP_TOPICS_COUNT] = {
        [0].str = custom_topic,
        [0].len = strlen(custom_topic),
        [1].str = custom_topic_2,
        [1].len = strlen(custom_topic_2)
    };

    err = aws_iot_subscription_topics_add(topics_list,
                          ARRAY_SIZE(topics_list));
    if (err) {
        printk("aws_iot_subscription_topics_add, error: %d\n", err);
    }

    return err;
}

static void date_time_event_handler(const struct date_time_evt *evt)
{
    switch (evt->type) {
    case DATE_TIME_OBTAINED_MODEM:
        /* Fall through */
    case DATE_TIME_OBTAINED_NTP:
        /* Fall through */
    case DATE_TIME_OBTAINED_EXT:
        printk("Date time obtained\n");
        k_sem_give(&date_time_obtained);

        /* De-register handler. At this point the sample will have
         * date time to depend on indefinitely until a reboot occurs.
         */
        date_time_register_handler(NULL);
        break;
    case DATE_TIME_NOT_OBTAINED:
        printk("DATE_TIME_NOT_OBTAINED\n");
        break;
    default:
```

```
        printk("Unknown event: %d", evt->type);
        break;
    }
}

void main(void)
{
    int err;

    printk("The AWS IoT sample started, version: %s\n", CONFIG_APP_VERSION);

    cJSON_Init();

#if defined(CONFIG_NRF_MODEM_LIB)
    nrf_modem_lib_dfu_handler();
#endif

    err = aws_iot_init(NULL, aws_iot_event_handler);
    if (err) {
        printk("AWS IoT library could not be initialized, error: %d\n",
               err);
    }

    /** Subscribe to customizable non-shadow specific topics
     *  to AWS IoT backend.
     */
    err = app_topics_subscribe();
    if (err) {
        printk("Adding application specific topics failed, error: %d\n",
            err);
    }

    work_init();
#if defined(CONFIG_NRF_MODEM_LIB)
    modem_configure();

    err = modem_info_init();
    if (err) {
        printk("Failed initializing modem info module, error: %d\n",
            err);
    }

    k_sem_take(&lte_connected, K_FOREVER);
#endif

    /* Trigger a date time update. The date_time API is used to timestamp data that is sent
     * to AWS IoT.
     */
    date_time_update_async(date_time_event_handler);
```

```
    /* Postpone connecting to AWS IoT until date time has been obtained. */
    k_sem_take(&date_time_obtained, K_FOREVER);
    k_work_schedule(&connect_work, K_NO_WAIT);
}
```

Lisa 2 - ENS210 kood kasutades SDK 2.1.0 draivereid

```c
#include <zephyr/zephyr.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/sys/printk.h>

void main(void)
{
        const struct device *dev = DEVICE_DT_GET_ONE(ams_ens210);
        struct sensor_value temperature, humidity;

        if (!device_is_ready(dev)) {
                printk("Device %s is not ready\n", dev->name);
                return;
        }

        printk("device is %p, name is %s\n", dev, dev->name);

        while (1) {
                sensor_sample_fetch(dev);
                sensor_channel_get(dev, SENSOR_CHAN_HUMIDITY, &humidity);
                sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temperature);
                printk("Temperature: %d.%06d C; Humidity: %d.%06d%%\n",
                        temperature.val1, temperature.val2,
                        humidity.val1, humidity.val2);

                k_sleep(K_MSEC(1000));
        }
}
```

Lisa 3 - HC-SR04 kood [34]

```
#include <stdbool.h>
#include <stdint.h>

#include <zephyr/zephyr.h>
#include <zephyr/drivers/gpio.h>
#include <zephyr/sys/printk.h>

#define LED0_NODE DT_ALIAS(led0)
#define LED1_NODE DT_ALIAS(led1)
#define CC_USED 0

/* Sisendi ja väljundi määramine led0 ja led1 alla */
static const struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);

/* Tameri seadistamine */
void timer_setup(void)
{
    NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_32Bit;
    NRF_TIMER1->SHORTS = TIMER_SHORTS_COMPARE0_CLEAR_Enabled <<
TIMER_SHORTS_COMPARE0_CLEAR_Pos| TIMER_SHORTS_COMPARE0_STOP_Enabled <<
TIMER_SHORTS_COMPARE0_STOP_Pos;
    NRF_TIMER1->PRESCALER = 0;
}

void timer_sleep(uint32_t timeout_us)
{

    if (timeout_us == 0)
    {
        return;
    }
    if (timeout_us == 1)
    {
        /* Mikrosekundi kalibreerimine vastava koguse _NOP() arvuga */
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP();
        return;
    }

    NRF_TIMER1->CC[CC_USED] = (timeout_us << 4) - 18;
```

```
    NRF_TIMER1->TASKS_START = 1;
    while(NRF_TIMER1->EVENTS_COMPARE[CC_USED] == 0)
    {
      /* Call __WFE() if loop is long */
    }
    NRF_TIMER1->EVENTS_COMPARE[CC_USED] = 0;
}

int main(void)
{
    NRF_CLOCK->TASKS_HFCLKSTART = 1;
    while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0);
    NRF_NVMC->ICACHECNF = NVMC_ICACHECNF_CACHEEN_Msk;

    timer_setup();

    int trigPin;
    int echoPin;
    int duration = 0;
    int distance;
    int constant;

    if (!device_is_ready(led0.port)) {
        return;
    }
    /* Sisendid ja väljundid */
    trigPin = gpio_pin_configure_dt(&led0, GPIO_OUTPUT_ACTIVE);
    echoPin = gpio_pin_configure_dt(&led1, GPIO_INPUT);
    if (trigPin < 0) {
        return;
    }

    while (1) {
        /* Annan impulsi 10 mikrosekundit*/
        gpio_pin_set_dt(&led0,0); //trig pin 0
        timer_sleep(20);
        gpio_pin_set_dt(&led0,1); //trig pin 1
        timer_sleep(10);
        gpio_pin_set_dt(&led0,0); //trig pin 0
        timer_sleep(20);
        /* Ootan kuni impulss jõuab kohale */
        while (gpio_pin_get_dt(&led1) == 0){
            duration = 0;
        }
        /* Annan uue impulsi 10 mikrosekundit*/
        gpio_pin_set_dt(&led0,0); //trig pin 0
        timer_sleep(20);
        gpio_pin_set_dt(&led0,1); //trig pin 1
        timer_sleep(10);
```

```
        gpio_pin_set_dt(&led0,0); //trig pin 0
        /* Hakkan loendama aega kuni sisent led1 on 1 (HIGH), ehk saadetud impulss
lülitab echo välja*/
        while (gpio_pin_get_dt(&led1) == 1){
            duration++;
            timer_sleep(1);
        }
        /* Arvestame veaga mis võib tekkida määtmisel ja arvutame vahemaa ümber*/
        constant = duration / 2;
        duration = (constant * 1) + duration;
        /* Arvutan aja millal echo ehk led1 sisend oli 1 (HIGH) */
        distance = duration*0.034/2;

        printk("-----\n");
        printk("Loendur %d mikrosekundit\n", duration);
        printk("echoPin %d\n", echoPin);
        printk("Kaugus %d cm\n", distance);
        k_msleep(500);
    }
}
```

Lisa 4 - ToF anduri VL53L0X kood kasutades SDK 2.1.0 draivereid

```
#include <zephyr/zephyr.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <stdio.h>
#include <zephyr/sys/printk.h>

void main(void)
{
    const struct device *dev = DEVICE_DT_GET_ONE(st_vl53l0x);
    struct sensor_value value;
    int ret;

    if (!device_is_ready(dev)) {
        //I2C andur ei ole liinil
        printk("TOF device not ready.\n");
        return;
    }

    while (1) {
        ret = sensor_sample_fetch(dev);
        if (ret) {
            //Näidu võtmine ebaõnnestus
            printk("Sensor sample fetch failed ret %d\n", ret);
            return;
        }

        ret = sensor_channel_get(dev, SENSOR_CHAN_DISTANCE, &value);
        // Kaugus meetrites
        printf("Distance is %d.%06d m\n", value.val1, value.val2);

        k_sleep(K_MSEC(500));
    }
}
```

Lisa 5 - Vingugaasianduri kood [32] [33]

```
#include <zephyr.h>
#include <device.h>
#include <sys/printk.h>
#include <drivers/gpio.h>
#include <drivers/adc.h>
#include <string.h>
#include <math.h>
/*ADC kanali parameetrid*/
#include <hal/nrf_saadc.h>
#define ADC_DEVICE_NODE     DT_INST(0, nordic_nrf_saadc)
#define ADC_RESOLUTION      10
#define ADC_GAIN            ADC_GAIN_1_5
#define ADC_REFERENCE       ADC_REF_INTERNAL
#define ADC_ACQUISITION_TIME    ADC_ACQ_TIME(ADC_ACQ_TIME_MICROSECONDS, 10)
#define ADC_CHANNEL_ID  0
#define ADC_CHANNEL_INPUT   NRF_SAADC_INPUT_AIN1
#define TIMER_INTERVAL_MSEC 1000
#define BUFFER_SIZE 1


struct k_timer my_timer;
const struct device *adc_dev;

static const struct adc_channel_cfg m_1st_channel_cfg = {
    .gain = ADC_GAIN,
    .reference = ADC_REFERENCE,
    .acquisition_time = ADC_ACQUISITION_TIME,
    .channel_id = ADC_CHANNEL_ID,
#if defined(CONFIG_ADC_CONFIGURABLE_INPUTS)
    .input_positive = ADC_CHANNEL_INPUT,
#endif

};


static int16_t m_sample_buffer[BUFFER_SIZE];

void adc_sample_event(struct k_timer *timer_id){

        float sensor_volt;
        float rs_gas;
        float r0;
        float r2 = 10400;
        int err;
        const struct adc_sequence sequence = {
        .channels    = BIT(ADC_CHANNEL_ID),
        .buffer      = m_sample_buffer,
```

```
        .buffer_size = sizeof(m_sample_buffer),
        .resolution  = ADC_RESOLUTION,
    };

    err = adc_read(adc_dev, &sequence);
    if (err) {
        printk("adc_read() failed with code %d\n", err);
    }

    printk("ADC value:");
    for (int i = 0; i < BUFFER_SIZE; i++) {

        float value = (m_sample_buffer[i]);

        sensor_volt=value/1024*3.3; //pinge väljundis
        rs_gas =r2 * (3.3-sensor_volt)/sensor_volt; //anduri takistus
        float ratio = rs_gas/r2;
        float lgppm = (log10(ratio) * -3.7)+ 0.9948;
        float ppm = pow(10,lgppm);
        printf("%3.2f\n ppm ", ppm);

    }
    printk("\n");
}

void main(void)
{
    int errAdc;
    //ADC häälestus
    adc_dev = DEVICE_DT_GET(ADC_DEVICE_NODE);
    if (!adc_dev) {
        printk("device_get_binding ADC failed\n");
    }

    errAdc = adc_channel_setup(adc_dev, &m_1st_channel_cfg);
    if (errAdc) {
        printk("Error in adc channel1 setup: %d\n", errAdc);
    }
    //Timeri häälestus
    k_timer_init(&my_timer, adc_sample_event, NULL);
    k_timer_start(&my_timer, K_MSEC(TIMER_INTERVAL_MSEC), K_MSEC(TIMER_INTERVAL_MSEC));
}
```

Lisa 6 - AWS kood [32] [33] [34]

```
#include <zephyr/kernel.h>
#include <stdio.h>
#include <stdlib.h>

/* Modem ja LTE */
#include <modem/lte_lc.h>
#include <modem/nrf_modem_lib.h>
#include <modem/modem_info.h>
#include <nrf_modem.h>

/* AWS IoT */
#include <net/aws_iot.h>

/* JSON */
#include <cJSON.h>
#include <cJSON_os.h>

/* ENS210 ja TOF */
#include <zephyr/zephyr.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/sys/printk.h>

/* HC-SR04 GPIO */
#include <zephyr/drivers/gpio.h>
#include <stdbool.h>
#include <stdint.h>

/* CO  */
#include <drivers/adc.h>
#include <string.h>
#include <math.h>
#include <hal/nrf_saadc.h>

/* HC-SR04 led0 ja led1 defineerimine */
#define LED0_NODE DT_ALIAS(led0)
#define LED1_NODE DT_ALIAS(led1)
#define CC_USED 0

/* CO ADC kanali parameetrite defineerimine */
#define ADC_DEVICE_NODE     DT_INST(0, nordic_nrf_saadc)
#define ADC_RESOLUTION      10
#define ADC_GAIN        ADC_GAIN_1_5
#define ADC_REFERENCE       ADC_REF_INTERNAL
#define ADC_ACQUISITION_TIME    ADC_ACQ_TIME(ADC_ACQ_TIME_MICROSECONDS, 10)
#define ADC_CHANNEL_ID  0
#define ADC_CHANNEL_INPUT   NRF_SAADC_INPUT_AIN1
```

```c
#define BUFFER_SIZE 1

/* CO ADC kanali häälestus */
const struct device *adc_dev;

static const struct adc_channel_cfg m_1st_channel_cfg = {
    .gain = ADC_GAIN,
    .reference = ADC_REFERENCE,
    .acquisition_time = ADC_ACQUISITION_TIME,
    .channel_id = ADC_CHANNEL_ID,
#if defined(CONFIG_ADC_CONFIGURABLE_INPUTS)
    .input_positive = ADC_CHANNEL_INPUT,
#endif

};

static int16_t m_sample_buffer[BUFFER_SIZE];

/* HC-SR04 I/O sidumine */
static const struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);

/* AWS IoT */
static struct k_work_delayable shadow_update_work;
static struct k_work_delayable connect_work;
static bool cloud_connected;

/* LTE */
static K_SEM_DEFINE(lte_connected, 0, 1);

NRF_MODEM_LIB_ON_INIT(aws_iot_init_hook, on_modem_lib_init, NULL);

/* Algväärtuse omistamine, hiljem 0 */
static int modem_lib_init_result = -1;

/* ENS210 anduri sidumine */
const struct device *dev = DEVICE_DT_GET_ONE(ams_ens210);
struct sensor_value temperature, humidity;

/* TOF anduri sidumine */
const struct device *dev2 = DEVICE_DT_GET_ONE(st_vl53l0x);
struct sensor_value value;

/* Modem */
static void on_modem_lib_init(int ret, void *ctx)
{
    modem_lib_init_result = ret;
}
```

```c
/* JSON objektid */
static int json_add_obj(cJSON *parent, const char *str, cJSON *item)
{
    cJSON_AddItemToObject(parent, str, item);

    return 0;
}

static int json_add_str(cJSON *parent, const char *str, const char *item)
{
    cJSON *json_str;

    json_str = cJSON_CreateString(item);
    if (json_str == NULL) {
        return -ENOMEM;
    }

    return json_add_obj(parent, str, json_str);
}

static int json_add_number(cJSON *parent, const char *str, double item)
{
    cJSON *json_num;

    json_num = cJSON_CreateNumber(item);
    if (json_num == NULL) {
        return -ENOMEM;
    }

    return json_add_obj(parent, str, json_num);
}

/* AWS IoT topiku sisu uunedamine saadud andutite väärtustega hetkes ja
kirjutamine JSON formaati */
static int shadow_update()
{
    int err;
    char *message;
    /* HC-SR04 muutujad */
    int counter;
    int trigPin;
    int echoPin;
    int duration = 0;
    float distance;
    int constant;
    /* CO muutujad */
    float sensor_volt;
    float rs_gas;
    float r2 = 10400; /* R2 takistuse väärtus anduri väljundis */
```

```
int coErr;
float ppm;

cJSON *root_obj = cJSON_CreateObject();
cJSON *state_obj = cJSON_CreateObject();
cJSON *reported_obj = cJSON_CreateObject();

if (root_obj == NULL || state_obj == NULL || reported_obj == NULL) {
    cJSON_Delete(root_obj);
    cJSON_Delete(state_obj);
    cJSON_Delete(reported_obj);
    err = -ENOMEM;
    return err;
}

/* CO ADC kanali häälestus/validerimine */
    const struct adc_sequence sequence = {
    .channels    = BIT(ADC_CHANNEL_ID),
    .buffer      = m_sample_buffer,
    .buffer_size = sizeof(m_sample_buffer),
    .resolution  = ADC_RESOLUTION,
};

coErr = adc_read(adc_dev, &sequence);
if (coErr) {
    printk("adc_read() failed with code %d\n", coErr);
}
/* CO väärtuse arvutamine */
printk("CO value:");
for (int i = 0; i < BUFFER_SIZE; i++) {
    float coValue = (m_sample_buffer[i]);
    sensor_volt=coValue/1024*3.3;
    rs_gas =r2 * (3.3-sensor_volt)/sensor_volt;
    float ratio = rs_gas/r2;
    float lgppm = (log10(ratio) * -3.7)+ 0.9948;
    ppm = pow(10,lgppm);
    printf("%3.2f\n ppm ", ppm);
}

/* HC-SR04 */
/* Annan impulsi 10 mikrosekundit*/
gpio_pin_set_dt(&led0,0); //trig pin 0
timer_sleep(20);
gpio_pin_set_dt(&led0,1); //trig pin 1
timer_sleep(10);
gpio_pin_set_dt(&led0,0); //trig pin 0
timer_sleep(20);
/* Ootan kuni impulss jõuab kohale */
while (gpio_pin_get_dt(&led1) == 0){
```

```
        duration = 0;
    }
    /* Annan uue impulsi 10 mikrosekundit*/
    gpio_pin_set_dt(&led0,0); //trig pin 0
    timer_sleep(20);
    gpio_pin_set_dt(&led0,1); //trig pin 1
    timer_sleep(10);
    gpio_pin_set_dt(&led0,0); //trig pin 0
    /* Hakkan loendama aega kuni sisent led1 on 1 (HIGH), ehk saadetud impulss lülitab
echo välja*/
    while (gpio_pin_get_dt(&led1) == 1){
        duration++;
        timer_sleep(1);
    }
    /* Arvestame veaga mis tekib kestvuse määtmisel */
    constant = duration / 2;
    duration = (constant * 1) + duration;
    /* Arvutan aja millal echo ehk led1 sisend oli 1 (HIGH) */
    distance = duration*0.034/2;
    printk("Ultrasonic distance %d cm\n", distance);


    /* ENS210 anduri väärtuste omistamine muutujatele temp ja humid*/
    sensor_sample_fetch(dev);
    sensor_channel_get(dev, SENSOR_CHAN_HUMIDITY, &humidity);
    sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temperature);
    printk("Temperature: %d.%06d C; Humidity: %d.%06d%%\n",
        temperature.val1, temperature.val2,
        humidity.val1, humidity.val2);

    float temp= (temperature.val1+(temperature.val2 * 0.000001));
    float humid= (humidity.val1+(humidity.val2 * 0.000001));

    /* TOF anduri väärtuste lugemine ja kirjutamine muutujale tofDistance */
    int ret;
    ret = sensor_sample_fetch(dev2);
    if (ret) {
        printk("Sensor fetch failed ret %d\n", ret);
        return;
    }
    ret = sensor_channel_get(dev2, SENSOR_CHAN_DISTANCE, &value);
    printf("ToF Distance is %d.%06d m\n", value.val1, value.val2);
    double tofDistance= (value.val1+(value.val2 * 0.000001));

/* Väärtuste kirjutamine JSON formaati */
    err += json_add_number(reported_obj, "carbonMonoxide", ppm);
    err += json_add_number(reported_obj, "ultrasonicDistance", distance);
    err += json_add_number(reported_obj, "tofDistance", tofDistance);
    err += json_add_number(reported_obj, "temperature", temp);
```

```
    err += json_add_number(reported_obj, "humidity", humid);
    err += json_add_obj(state_obj, "values", reported_obj);
    err += json_add_obj(root_obj, "temp_hum", state_obj);

    if (err) {
        printk("json_add, error: %d\n", err);
        goto cleanup;
    }

    message = cJSON_Print(root_obj);
    if (message == NULL) {
        printk("cJSON_Print, error: returned NULL\n");
        err = -ENOMEM;
        goto cleanup;
    }

    /* Loodud andmete edastamine AWS IoT topiku
    $aws/things/temp_hum/shadow/update*/
    struct aws_iot_data tx_data = {
        .qos = MQTT_QOS_0_AT_MOST_ONCE,
        .topic.type = AWS_IOT_SHADOW_TOPIC_UPDATE,
        .ptr = message,
        .len = strlen(message)
    };

    printk("Publishing: %s to AWS IoT broker\n", message);

    err = aws_iot_send(&tx_data);
    if (err) {
        printk("aws_iot_send, error: %d\n", err);
    }

    cJSON_FreeString(message);

    cleanup:

    cJSON_Delete(root_obj);

    return err;
}

/* AWS IoT ühenduse kontrollimine ja kui ühendust ei ole millal proovitakse
uuesti ühendust saada */
static void connect_work_fn(struct k_work *work)
{
    int err;

    if (cloud_connected) {
        return;
```

```
    }

    err = aws_iot_connect(NULL);
    if (err) {
        printk("aws_iot_connect, error: %d\n", err);
    }

    printk("Next connection retry in %d seconds\n",
           CONFIG_CONNECTION_RETRY_TIMEOUT_SECONDS);

    k_work_schedule(&connect_work,
            K_SECONDS(CONFIG_CONNECTION_RETRY_TIMEOUT_SECONDS));
}

/* AWS topiku teema publitseerimine, kui õnnestus siis millal uus edastus toimub
ja kui ei õnnestunud siis viga */
static void shadow_update_work_fn(struct k_work *work)
{
    int err;

    if (!cloud_connected) {
        return;
    }

    err = shadow_update(false);
    if (err) {
        printk("shadow_update, error: %d\n", err);
    }

    printk("Next data publication in %d seconds\n",
           CONFIG_PUBLICATION_INTERVAL_SECONDS);

    k_work_schedule(&shadow_update_work,
            K_SECONDS(CONFIG_PUBLICATION_INTERVAL_SECONDS));
}


/* AWS IoT erinevate juhtumite tegevused */
void aws_iot_event_handler(const struct aws_iot_evt *const evt)
{
    switch (evt->type) {
    case AWS_IOT_EVT_CONNECTING:
        printk("AWS_IOT_EVT_CONNECTING\n");
        break;
    case AWS_IOT_EVT_CONNECTED:
        printk("AWS_IOT_EVT_CONNECTED\n");

        cloud_connected = true;
        (void)k_work_cancel_delayable(&connect_work);
```

```
        /** Alustatakse topiku uuendamist etteantud aja jooksul (60 sek).
         */
        k_work_schedule(&shadow_update_work,
                K_SECONDS(CONFIG_PUBLICATION_INTERVAL_SECONDS));

    #if defined(CONFIG_NRF_MODEM_LIB)
        int err = lte_lc_psm_req(true);
        if (err) {
            printk("Requesting PSM failed, error: %d\n", err);
        }
    #endif
        break;
    case AWS_IOT_EVT_READY:
        printk("AWS_IOT_EVT_READY\n");
        break;
    case AWS_IOT_EVT_DISCONNECTED:
        printk("AWS_IOT_EVT_DISCONNECTED\n");
        cloud_connected = false;
        (void)k_work_cancel_delayable(&shadow_update_work);
        k_work_schedule(&connect_work, K_NO_WAIT);
        break;
    }

}

static void work_init(void)
{
    k_work_init_delayable(&shadow_update_work, shadow_update_work_fn);
    k_work_init_delayable(&connect_work, connect_work_fn);

}

/* LTE võrguga ühendamine*/
#if defined(CONFIG_NRF_MODEM_LIB)
static void lte_handler(const struct lte_lc_evt *const evt)
{
    switch (evt->type) {
    case LTE_LC_EVT_NW_REG_STATUS:
        if ((evt->nw_reg_status != LTE_LC_NW_REG_REGISTERED_HOME) &&
             (evt->nw_reg_status != LTE_LC_NW_REG_REGISTERED_ROAMING)) {
            break;
        }

        printk("Network registration status: %s\n",
            evt->nw_reg_status == LTE_LC_NW_REG_REGISTERED_HOME ?
            "Connected - home network" : "Connected - roaming");

        k_sem_give(&lte_connected);
```

```
        break;
    case LTE_LC_EVT_PSM_UPDATE:
        printk("PSM parameter update: TAU: %d, Active time: %d\n",
            evt->psm_cfg.tau, evt->psm_cfg.active_time);
        break;
    case LTE_LC_EVT_EDRX_UPDATE: {
        char log_buf[60];
        ssize_t len;

        len = snprintf(log_buf, sizeof(log_buf),
                   "eDRX parameter update: eDRX: %f, PTW: %f",
                   evt->edrx_cfg.edrx, evt->edrx_cfg.ptw);
        if (len > 0) {
            printk("%s\n", log_buf);
        }
        break;
    }
    case LTE_LC_EVT_RRC_UPDATE:
        printk("RRC mode: %s\n",
            evt->rrc_mode == LTE_LC_RRC_MODE_CONNECTED ?
            "Connected" : "Idle");
        break;
    case LTE_LC_EVT_CELL_UPDATE:
        printk("LTE cell changed: Cell ID: %d, Tracking area: %d\n",
            evt->cell.id, evt->cell.tac);
        break;
    default:
        break;
    }
}

/* Modemi häälestamine */
static void modem_configure(void)
{
    int err;
    if (IS_ENABLED(CONFIG_LTE_AUTO_INIT_AND_CONNECT)) {
        /* Ära tee midagi kui modem häälestatud ja LTE ühendatud */
    } else {
        err = lte_lc_init_and_connect_async(lte_handler);
        if (err) {
            printk("Modem could not be configured, error: %d\n",
                err);
            return;
        }
    }
}
#endif

/* Tameri seadistamine */
```

```
void timer_setup(void)
{
    NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_32Bit;
    NRF_TIMER1->SHORTS = TIMER_SHORTS_COMPARE0_CLEAR_Enabled <<
TIMER_SHORTS_COMPARE0_CLEAR_Pos| TIMER_SHORTS_COMPARE0_STOP_Enabled <<
TIMER_SHORTS_COMPARE0_STOP_Pos;
    NRF_TIMER1->PRESCALER = 0;
}

/* Mikrosekundi defineerimine */
void timer_sleep(uint32_t timeout_us)
{

    if (timeout_us == 0)
    {
        return;
    }
    if (timeout_us == 1)
    {
        /* Mikrosekundi kalibreerimine vastava koguse _NOP() arvuga */
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP(); __NOP(); __NOP();
        __NOP(); __NOP();
        return;
    }

    NRF_TIMER1->CC[CC_USED] = (timeout_us << 4) - 18;
    NRF_TIMER1->TASKS_START = 1;
    while(NRF_TIMER1->EVENTS_COMPARE[CC_USED] == 0)
    {
      /* kutsub __WFE() kui ”loop” on pikk */
    }
    NRF_TIMER1->EVENTS_COMPARE[CC_USED] = 0;
}

void main(void)
{
    int err;
    int errAdc;
    /* Prindime rakenduse versiooni */
    printk("The AWS IoT started, version: %s\n", CONFIG_APP_VERSION);

    /* JSON häälestus*/
    cJSON_Init();
```

```
/* CO häälestus ja kontroll */
adc_dev = DEVICE_DT_GET(ADC_DEVICE_NODE);
if (!adc_dev) {
    printk("device_get_binding ADC failed\n");
}

errAdc = adc_channel_setup(adc_dev, &m_1st_channel_cfg);
if (errAdc) {
    printk("Error in adc channel A0 setup: %d\n", errAdc);
}
/* CO häälestus ja kontrolli lõpp */

/* HC_SR04 kanali häälestuse kontroll ja taimer */
NRF_CLOCK->TASKS_HFCLKSTART = 1;
while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0);
NRF_NVMC->ICACHECNF = NVMC_ICACHECNF_CACHEEN_Msk;

timer_setup();

    if (!device_is_ready(led0.port)) {
    return;
}

gpio_pin_configure_dt(&led0, GPIO_OUTPUT_ACTIVE);
gpio_pin_configure_dt(&led1, GPIO_INPUT);
/* HC_SR04 häälestuse ja kontrolli lõpp */

/* ENS210 seadme tuvastamine I2C liinil */
if (!device_is_ready(dev)) {
    printk("Device %s is not ready\n", dev->name);
    return;
}

printk("device is %p, name is %s\n", dev, dev->name);

/* ENS210 seadme tuvastamise lõpp */

/* TOF seadme tuvastamine I2C liinil */
if (!device_is_ready(dev2)) {
    printk("TOF device not ready.\n");
    return;
}

printk("device is %p, name is %s\n", dev2, dev2->name);
/* TOF seadme tuvastamise lõpp */

/* AWS IoT häälestus*/
err = aws_iot_init(NULL, aws_iot_event_handler);
```

```
    if (err) {
        printk("AWS IoT library could not be initialized, error: %d\n",
               err);
    }

    /* Modemi häälestus*/
    work_init();
#if defined(CONFIG_NRF_MODEM_LIB)
    modem_configure();

    err = modem_info_init();
    if (err) {
        printk("Failed initializing modem info module, error: %d\n",
            err);
    }

    k_sem_take(&lte_connected, K_FOREVER);
#endif

    /* AWS IoT ühendus*/
    k_work_schedule(&connect_work, K_NO_WAIT);
}
```

Lisa 7 - Näidikute paneelid Grafanas