Maiju Lehtonen

Laiteriippumaton ohjelmointi sulautetuille järjestelmille

Insinööri

Tieto- ja viestintätekniikka

Kevät 2023

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Lehtonen Maiju

Työn nimi: Laiteriippumaton ohjelmointi sulautetuille järjestelmille

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: laiteriippumaton ohjelmointi, Zephyr RTOS, modulaarinen ohjelmointi, C

Opinnäytetyö tehtiin Etteplan Oyj:n toimeksiantona. Työn tavoitteena oli toteuttaa laiteriippumattomaan ohjelmointiin ohjeistus ja sitä tukeva esimerkki. Ohjeistuksen tulee sisältää ohjeita, miten sulautetun laitteen ohjelmakoodi tulee suunnitella ja toteuttaa laiteriippumattomuuden näkökulmasta.

Opinnäytetyön tietoperusta koostui pääosin eri verkkolähteistä. Työn käytännön osuuden valinnat perustuivat pääosin Zephyrin esimerkkisovelluksiin, joista otettiin mallia. Käytännön osuutena toimi esimerkkiohjelma, jonka vaatimuksena oli laiteriippumattomuus. Ohjelman tuli olla helposti siirrettävissä toiselle kehitysalustalle, sekä käytetty sensori tuli olla helposti vaihdettavissa toiseen samankaltaiseen sensoriin. Ohjelma toteutetaan Zephyr RTOS -reaaliaikakäyttöjärjestelmää hyödyntäen.

Työssä tutkittiin ensin laiteriippumattomuutta ja modulaarisuutta sekä niiden hyötyjä ja ongelmia. Sen jälkeen siirryttiin tutkimaan Zephyr-käyttöjärjestelmän ominaisuuksia ja käyttämistä. Esimerkkiohjelman tekeminen aloitettiin ensimmäisessä vaiheessa keskittymällä ohjelman siirtämiseen toiselle kehitysalustalle ja yhden anturin arvojen tulostamiseen terminaaliin. Toisessa vaiheessa keskityttiin säilyttämään ensimmäisessä vaiheessa saatu siirrettävyys, mutta lisättiin anturin vaihtaminen toiseen samankaltaiseen anturiin.

Työn tuloksena saatiin aikaiseksi vaatimukset täyttävä esimerkkisovellus, joka oli helposti siirrettävissä kehitysalustalta toiselle sekä kehitysalustassa oleva anturi oli helppo pienillä koodimuutoksilla ja uudella peittokuvalla vaihdettavissa. Itse työ toimi vaadittuna ohjeistuksena yritykselle.

Johtopäätöksenä opinnäytetyö antaa hyvän lähtökohdan laiteriippumattomaan ohjelmointiin sulautetulle järjestelmälle ja todentaa laiteriippumattoman ohjelmoinnin käyttämisen toimivuuden. Opinnäytetyötä seuratessa saa käsityksen yhdestä mahdollisesta laiteriippumattomuutta tukevasta käyttöjärjestelmästä, jonka käyttö teollisuudessa voi yleistyä lähitulevaisuudessa. Zephyr tarjoaa paljon mahdollisuuksia laiteriippumattomuudelle ja tämä työ esittää vain murto-osan siitä.

KAMK • University of Applied Sciences

Abstract

Author: Lehtonen Maiju

Title of the Publication: Device Independent Programming for Embedded Devices

Degree Title: Bachelor of Engineering, Information technology

Keywords: device independent programming, Zephyr RTOS, modular programming, C

The thesis was commissioned by Etteplan Oyj. The goal of the work was to implement instructions and an example application for device-independent programming. The instructions must include instructions on how the program code for the embedded device should be designed and implemented from the point of device independence.

The database of the thesis mainly consisted of various online sources. The choices for the practical part of the work were mainly based on Zephyr's example applications. The practical part served as an example tutorial, the requirement of which was device independence. The program must be easily transferable to another development platform, and the used sensor must be easily replaceable with another similar sensor. The program is implemented using the Zephyr RTOS real-time operating system.

The work first investigated device independence and modularity, as well as their benefits and problems. After that, the work moved on to studying the features and usage of the Zephyr operating system. Making the example program was started in the first phase by focusing on transferring the program to another development platform and printing the values of a sensor to the terminal. In the second phase, the focus was on maintaining the portability obtained in the first phase and the replacement of the sensor with another similar sensor was added.

As a result of the work, an example application that met the requirements was completed, which could easily be transferred from one development platform to another, and the sensor in the development platform could easily be replaced with small code changes and a new overlay. The thesis itself served as the required instructions for the company.

In conclusion, the thesis provides a good starting point for a system embedded in device-independent programming and verifies the functionality of using device-independent programming. The thesis gives an idea of one possible operating system that supports device independence, the use of which may become more common in the industry soon. Zephyr offers many possibilities for device-freedom, and this work presents only a fraction of it.

Sisällys

Symboliluettelo

C	Ohjelmointikieli
Nordic nfr52480	Kehitysalusta
STnucleo f334r8	Kehitysalusta
Zephyr RTOS	Reaaliaikainen käyttöjärjestelmä

1 Johdanto

Työn aihe tulee Etteplan Oyj:ltä. Työssä tutkitaan, miten kehitetään laiteriippumaton ohjelma Zephyr RTOS -reaaliaikakäyttöjärjestelmää hyödyntäen. Aihe sai alkunsa komponenttipulasta. Silloin yrityksessä alettiin miettiä, miten puuttuvista komponenteista kärsiviä laitteita saataisiin silti edistettyä toisilla komponenteilla.

Opinnäytetyön tavoitteena oli luoda sulautettuja järjestelmiä kehittävälle yritykselle ohjeistus ja sitä tukevat esimerkit, miten sulautetun laitteen ohjelmakoodi tulee suunnitella ja toteuttaa laiteriippumattomuuden näkökulmasta. Yrityksen jo olemassa olevat käytännöt sekä laite- ja ohjelmistokomponenttivalinnat tuli huomioida myös opinnäytetyön yhteydessä. Ohjeistuksen kohderyhmänä toimivat yrityksen sulautetun ohjelmiston kehittäjät, joilla on vaihteleva tuntemus laiteläheisen ohjelmoinnin, abstraktiokerrosten sekä reaaliaikakäyttöjärjestelmien käytöstä.

Opinnäytetyön sisällön tavoitteena oli tukea, että laiteriippumattoman ohjelmistokehityksen perusperiaatteita noudatetaan käytännön ohjelmistokehityksessä. Opinnäytetyö tulee myöhemmin yrityksen ohjelmistokehitysprosessin tukimateriaaliksi.

Opinnäytetyö koostuu taustatiedoista ja -oletuksista, kokeellisesta käytännön osuudesta ja näistä myöhemmin jalostettavasta ohjeistuksesta käytännön esimerkkeineen. Ohjeistus on osa opinnäytetyötä. Käytännön osuuden tavoitteena on testata dokumentissa mainittuja oletuksia ja hyviä sovelluskehitysmenetelmiä laitteistolla, joka valittiin vastaamaan usein yrityksen asiakasprojekteissa käytettyjä. Lopuksi on tarkoitus koostaa kokeilusta saatujen lopputulosten perusteella sekä esimerkit että ohjeistus.

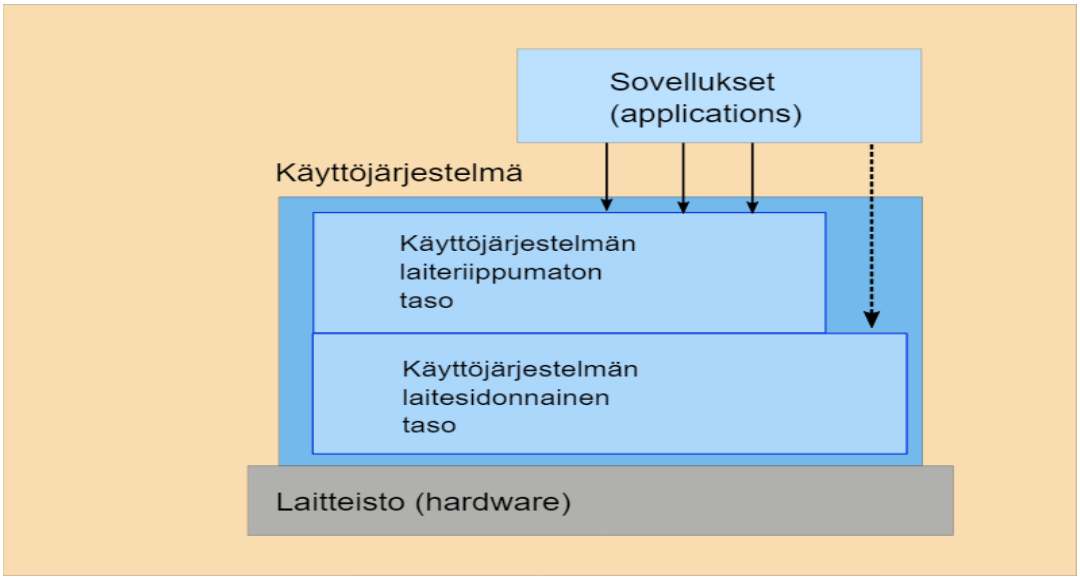
2 Laiteriippumaton ohjelma ja laiteriippumaton ohjelmointi

Laiteriippumattomuus on prosessi, joka tuottaa ohjelman, jota pystytään käyttämään hyvin usealla laitteella riippumatta laitteen komponenteista [1]. Laiteriippuvaiset ohjelman osat tehdään jollain muulla ohjelmalla tai ohjelmaan lisättävillä ajureilla.

2.1 Laiteriippumattomuus sulautetuilla järjestelmillä

Laiteriippumattomuus sulautetuilla järjestelmillä luodaan pääsääntöisesti käyttöjärjestelmävalinnalla. Eri käyttöjärjestelmät tukevat laiteriippumattomuutta eritasoisesti. Esimerkiksi Zephyr RTOS -käyttöjärjestelmä tukee hyvin montaa eri kehitysalustaa ja hyödyntää laitepuuta sovelluksen rakentamisessa. Molemmat ominaisuudet tukevat laiteriippumattoman ohjelman ohjelmointia.

Laiteriippumattomuutta lisätään laiteajureilla ja laitepuulla. Laitepuu määrittelee pinnit ja niihin kytketyt laitteet. Näin itse ohjelman ei tarvitse muuttua laitteelta toiselle, ainoastaan laitepuun tulee muuttua. Ohjelma ja käyttöjärjestelmä yhdessä tuottavat laiteriippumattoman ohjelman, kuten kuvassa 1 esitetään. Ohjelma käsittelee käyttöjärjestelmän laiteriippumatonta tasoa. Ohjelman avuksi tuodaan laitepuu, joka kertoo ohjelmalle, mikä laite ja miten se on kytketty laitteistoon. Laitepuu käsittelee käyttöjärjestelmän laitesidonnaista tasoa. Ohjelma ja laitepuu muodostavat yhdessä sovelluksen, joka on helposti siirrettävä toiselle laitteistolle. Siirtoa varten täytyy ainoastaan luoda uusi laitepuu, joka kertoo uuden laitteiston kytkennän ohjelmalle.



Kuva 1. Kuvaus laiteriippumattomasta ohjelmasta [1].

2.2 Laiteriippumattoman ohjelmoinnin hyödyt ja huonot puolet

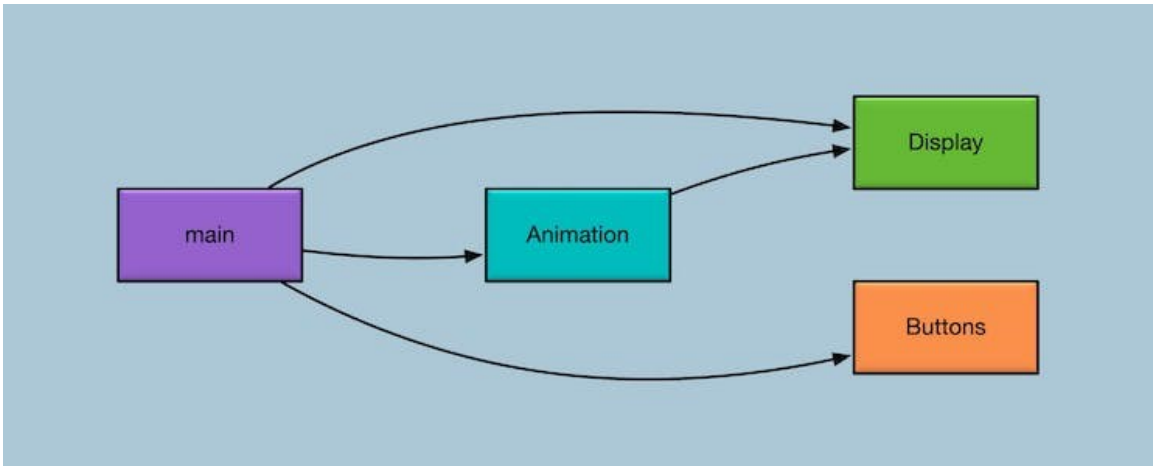
Laiteriippumattomuus antaa yrityksille paljon hyötyä. Ohjelmat toimivat kaikilla laitteilla, joten yrityksellä on paljon vapautta laitteiden suhteen. Ohjelmakehitys on myös vapaampaa, koska ei tarvitse miettiä, mille laitteelle ohjelma tulee. [2.] Laiteriippumattomuus antaa myös liikkumavaraa laitteen tai sovelluksen suunnittelussa. Jos jokin osa on loppunut tai siinä on hyvin pitkät toimitusajat, on laite helppo vaihtaa toiseen ilman suurempia lisätöitä. Laiteriippumattomuus antaa myös mahdollisuuden kehittää sovellus haluttuja ominaisuuksia miettien eikä laitteen rajoituksien mukaan.

Laiteriippumaton ohjelma voi vaatia ylimääräistä koodia, jotka laiteriippuvaisessa ohjelmassa tulisivat laitemääritysien tai laitteen kansiorakenteen kautta. Ohjelmia joudutaan myös ehkä yksinkertaistamaan, jotta se sopii myös vähemmän tehokkaille laitteille. [3.]

Laiteriippumattomuudessa joudutaan myös rajoittamaan, miten paljon laiteriippumattomuutta kannattaa käyttää. Sulautetuille järjestelmille laiteriippumattomuutta ei todennäköisesti kannata kokonaan tehdä, sillä alustat vaativat jonkin verran laiteriippuvaista koodia toimiakseen. Laiteriippumattomuutta voi kuitenkin hyödyntää enemmissä määrin, ja näin hyödyntää samaa koodin osaa monessa eri laitteessa. Tämä vähentää koodin tekoa ja nopeuttaa laitekehitystä.

3 Modulaarisuus ohjelmoinnissa

Modulaarinen ohjelma on rakennettu monesta omasta moduulista, jotka yhdessä tekevät kokonaisen ohjelman. Ohjelman moduulit toimivat omina osuuksinaan ja omina tiedostoinaan, jotka kerätään toimimaan yhdessä pääohjelmassa. Moduulien ei tulisi välittää, mitä muita moduuleita ohjelmassa on, keskittyä ainoastaan omaan toimintaansa. Pääohjelman täytyy huolehtia siitä, miten moduuleita käytetään yhdessä. Esimerkkikuvaus modulaarisesta koodista kuvassa 2.



Kuva 2. Kuvaus modulaarisesta ohjelmasta [4].

Moduulin tulisi keskittyä yhteen asiaan, jolloin sitä on helpompi hallita pääohjelmassa. Moduulit tuovat ohjelmointiin myös nopeutta, koska samoja moduuleita voidaan hyödyntää useissa eri ohjelmissa. Moduulien ei tulisi olla riippuvainen toisista moduuleista tai pääohjelmasta. Tällä tavoin ei ohjelmoijan tarvitse kehittää moduulin asioita useasti moneen eri ohjelmaan. Ohjelman jakaminen pienempiin moduuleihin helpottaa myös ohjelman kehittämistä. Usea kehittäjä voi kehittää ohjelmaa yhdessä helposti, sillä jokainen voi keskittyä omaan moduuliinsa. Pienempiä moduuleita on myös paljon helpompi päivittää tarpeen vaatiessa ja koodia on helpompi lukea.

3.1 Laiteriippumaton ja modulaarinen ohjelma

Laiteriippumaton ja modulaarinen ohjelmointi sopivat hyvin yhteen. Modulaarisella koodilla voidaan pyrkiä kohti laiteriippumattomuutta eristämällä laiteriippuvaiset osat omaksi kokonaisuudekseen. Näin ollen suurin osa ohjelmasta voi olla laiteriippumatonta.

Laiteriippumattomuus ja modulaarisuus tulevat yhteen jo osittain käyttöjärjestelmän kautta. Esimerkiksi Zephyr RTOS vaatii ohjelman koodin lisäksi peittokuvan, jolla ohjelmalle kerrotaan, mitä kaikkea laitteessa on kiinni.

3.2 Modulaarisuuden hyödyt ja haitat

Modulaarisuus tuo ohjelmaan toimintavarmuutta, vähentää kehityskuluja ja on helpompi ylläpitää [5]. Modulaarisuudella saadaan myös uudelleen käytettyä koodin osia toisissa projekteissa, varsinkin, jos moduuli on laiteriippumaton. Moduulit ovat yksinkertaisempia ja lyhyempiä kuin kaiken pääohjelmaan kirjoittaminen, joten sitä on helpompi lukea. Modulaarista ohjelmaa tehdessä useampi henkilö voi osallistua ohjelman tekemiseen, sillä kaikki voivat kirjoittaa omia moduuleita ilman, että ne vaikuta toisen tekemiseen mitenkään. [6.]

Modulaarinen ohjelma voi olla raskaampi ajaa kuin ei-modulaarinen ohjelma. Tämä voi tuottaa ongelmia joillekin kehitysalustoille, mutta suurin osa kehitysalustoista on tarpeeksi tehokkaita käsittelemään modulaarisia ohjelmia [5].

4 Zephyr RTOS

Zephyr RTOS -käyttöjärjestelmä perustuu pienijalanjälkiseen kerneliin. Sitä käytetään niin vähäresurssisissa laitteissa kuin sulautetuissa laitteissa, ledien vilkuttelusta hyvin monimutkaisiin kontrollereihin, kuten älykelloihin. Zephyr RTOS tukee hyvin montaa arkkitehtuuria ja eri kehitysalustoja [7]. Zephyr RTOS on ilmainen vapaan lähdekoodin käyttöjärjestelmä, joten sitä on alettu suosia teollisuuden parissa enemmissä määrin.

4.1 Asennus ja käyttöönotto

Zephyrin käyttöönotto tapahtuu helposti seuraamalla Zephyrin omaa pikaopasta. Sen avulla saa tarvittavat kehitysympäristöt omalle koneelleen. Zephyrin voi asentaa joko Linux-, MacOS- tai Windows-käyttöjärjestelmälle. Asennus ja käyttäminen muuttuvat hieman käyttöjärjestelmän mukaan. [8.]

Kehitysympäristön asennuksen jälkeen jotkin kehitysalustat vaativat lisäosien asennusta. Lisäosat määräytyvät käytettävien kehitysalustoiden mukaan. Hyvin monelle eri kehitysalustalle löytyy oma quick start -opas. Opasta seuraamalla pystyy testaamaan oman kehitysalustan ja saada esimerkin, miten kehitysalustalle kehitetään omia ohjelmia.

4.2 Käytettävä kansiorakenne

Jokaiselle kehitettävälle sovellukselle tulee olla tietynlainen kansiorakenne, jotta ohjelman saa asennettua kehitysalustalle. Kansiorakenne perustuu CMake-ohjelmaan [9]. Sovellukset kannattaa asentaa Zephyrin ohjeiden mukaisesti käyttäjän zephyrproject kansion sisälle ”home/zephyrproject/yourappname” ja yourappname-kansio sisältää build-kansion. Build-kansio sisältää tarvittavat asiat sovelluksen rakentamiseen ja siirtämiseen kehitysalustalle. Tämän kansiorakenteen käyttäminen helpottaa West build -komennon käyttöä. Yourappname-kansion loppurakenne on kuvattu kuvassa 3.

```
<app>
├── CMakeLists.txt
├── app.overlay
├── prj.conf
└── src
    └── main.c
```

Kuva 3. Zephyr-kansiorakenne [9].

Kansio sisältää CMakeList.txt-tiedoston, joka kertoo, missä muut sovelluksen rakentamiseen tarvittavat tiedostot sijaitsevat. Tiedosto kertoo myös, missä rakennuksen aikana tarvittavat kehitysalustan linkit sijaitsevat. app.overlay-tiedosto on laitepuun peittokuva. Peittokuva kertoo kaikki komponentit, jotka ovat kytkettynä kehitysalustaan. Ilman peittokuvaa mitkään komponentit eivät tule toimimaan kehitysalustalla. prj.conf-tiedosto sisältää tarkat sovelluskohtaiset konfiguraatiotiedot. Konfiguraatiotiedot yhdistetään muiden asetusten kanssa rakennusvaiheessa. Konfiguraatio koskee yleisesti sovelluksen ominaisuuksia, kuten käytettävä tiedonsiirtoväylä tai virheenkorjauksen lokitiedot. src-kansio sisältää kaikki sovelluksen itsekirjoitetut ohjelmatiedostot. Osa laiteajureista ja muista Zephyrin sisältämistä funktioista löytyy Zephyrin omista tiedostoista. Näitä voidaan kuitenkin käyttää omassa ohjelmoinnissa. src-kansiossa oleva main.c-tiedosto sisältää pääohjelman, joka on yleisesti tiedostopäätteen mukaan kirjoitettu C:llä.

4.3 Uuden sovelluksen luominen

Zephyr suosittelee uuden sovelluksen tekemiseen heidän esimerkkisovelluspohjaansa. Pohja antaa suoraan tarvittavan kansiorakenteen ja tiedostot, joista on helppo lähteä rakentamaan omaa sovellusta. Esimerkkipohjan voi helposti ladata parilla komennolla. [9.] Esimerkkisovelluksen saa alustettua itselle komennoilla, jotka näkyvät kuvassa 4.

```
cd <home>/zephyrproject
git clone https://github.com/zephyrproject-rtos/example-application my-app
```

Kuva 4. Esimerkkisovelluksen alustaminen [9].

Oman sovelluksen nimeä voi helposti muokata vaihtamalla git clone -komennosta my-app haluamaksi sovelluksen nimeksi. Sovelluksen voi myös rakentaa itse valitsemalla tarvittavat tiedostot. Tähän monimutkaisempaan sovelluksen rakentamiseen löytyy ohjeistus Zephyrin sivuilta. [9.]

4.4 Laitepuun peittokuva

Zephyrissä täytyy luoda laitepuun peittokuva, johon esitellään kaikki pinneissä kiinni olevat komponentit. Varsinainen laitepuu on rakennettu kaikille tuetuille kehitysalustoille valmiiksi, mutta suurin osa sovelluksista tarvitsee vielä peittokuvan. Peittokuva nimetään kehitysalusta.overlay-nimellä, jotta ohjelman rakennusvaiheessa kääntäjä osaa valita peittokuvista juuri halutun kehitysalustan peittokuvan. Peittokuvia voi olla monta ohjelmaa kohti.

Peittokuva on tiedosto, joka kertoo ohjelmalle, missä pinnissä on minkäkinlainen laite kiinni, onko se tulo- vai lähtösuuntainen ja millä nimellä sitä voi kutsua. Nämä pinnit ovat jokaiselle kehitysalustalle erilaisia, joten täytyy löytää käytetylle kehitysalustalle sopivat laitepuun ohjaukset. [10.] Peittokuvan rakenne on kuvattu kuvassa 5.

```
    aliases {
        sw0 = &button0;
        sw1 = &button1;
    };
    gpio_keys {
        compatible = "gpio-keys";
        button0: button_0 {
            gpios = <&gpio0 0 GPIO_ACTIVE_LOW>;
            label = "Button 0";
        };
        button1: button_1 {
            gpios = <&gpio1 3 GPIO_ACTIVE_LOW>;
            label = "Button 1";
        };
    };
};
```

Kuva 5. Laitepuun peittokuvan rakenne [10].

Peittokuvan rakenne muodostuu aliases-osiosta, jossa kerrotaan, millä nimellä komponentteja voidaan ohjelmassa kutsua. Osio ei ole pakollinen, koska nimet voidaan myös antaa gpio_keys-osiossa. gpio_keys-osio sisältää tiedot komponenttien pinneistä ja komponentista itsessään.

gpio_keys voi kertoa, mitä default-pinnejä kehitysalustalla käytetään, joka nopeuttaa peittokuvan tekoa. Oletuspinnit löytyvät kehitysalustan laiteajurin tiedostoista. Osiossa voidaan myös alustaa laitteen tila.

5 Projektin toteutus

5.1 Alkutilanne ja tavoitteet

Projektin tavoitteena oli saada aikaiseksi toimiva demo Zephyr RTOS -käyttöjärjestelmällä laiteriippumattomasta ohjelmasta. Sovellus lukee anturia ja antaa siitä terminaaliin arvoja. Anturin tulee olla helposti vaihdettavissa toiseen samankaltaiseen anturiin ja koko sovelluksen tulee olla helposti siirrettävissä toiselle kehitysalustalle.

Työn käytännön osuuden ensimmäinen vaihe oli siinä käytettävien kehitysalustojen ja oheiskomponenttien valinta. Tuotteiden valintaperusteina oli paitsi Zephyr-reaaliaikakäyttöjärjestelmän yhteensopivuus ja jo olevassa oleva laiteajurituen saatavuus, mutta myös se, että komponentit vastaisivat mahdollisimman hyvin yrityksen asiakasprojekteissa useimmiten käytettyjä. Vaikka asiakasprojekteissa laitteiston valinnasta vastaavat usein yrityksen elektroniikkasuunnittelijat, ohjelmistosuunnittelijoiden vaikutusmahdollisuus ja -vastuu valintoihin on huomattava. Myös loppuasiakas voi suosia tiettyjä komponenttivalmistajia tai tuoteperheitä, joiden huomioonottaminen laitevalinnoissa on usein välttämätöntä.

5.2 Projektin suunnittelu ja suunnitelma

Projektin alkuperäinen suunnitelma oli aloittaa kehitystyö testaamalla, miten valitut osat toimivat Zephyrin kanssa. Alussa tehdään yksinkertainen esimerkkisovellus anturin arvojen lukemiseen, minkä jälkeen ohjelma yritetään siirtää toiselle kehitysalustalle. Siirron onnistuessa seuraavana vaiheena on yrittää vaihtaa anturi toiseen samankaltaiseen anturiin ja säilyttää silti ohjelman siirrettävyys alustalta toiselle.

Projektia suunnitellessa apuna käytettiin Zephyrin omia esimerkkisovelluksia. Esimerkkisovellusten perusteella sai käsityksen siitä, minkälainen ohjelmarakenne on tarpeellinen tehdä, jotta ohjelmasta tulisi laiteriippumaton. Sovellukset auttoivat myös oikeiden funktioiden löytämisessä, jotta sensorin arvoja saa haettua.

5.2.1 Käytetyt kehitysalustat

Merkittävä osa sulautettujen järjestelmien kehityksestä tapahtuu nykyään ARM Cortex M4 -pohjaisille mikrosuorittimille, joita eri komponenttivalmistajat tarjoavat. Näitä mikrosuorittimia käytetään hyväksi tässäkin työssä.

Ranskalais-italialainen ST Microelectronics on yksi tunnetuimmista valmistajista, laajalla valikoimallaan ja hyvällä ohjelmistokehitystuellaan [11]. Alustaksi opinnäytetyön käytännön vaiheeseen valikoitui asiakasprojektien laitteistoa hyvin vastaava ST Microelectronicsin STM32F3 -pohjainen Nucleo 334R8 -kehitysalusta.

Norjalainen Nordic Semiconductor on hyvin tunnettu kehittäjä ja valmistaja vähävirtaisten langattomien viestintäteknologian komponenttien parissa [12]. Vertailualustaksi työn käytännönvaiheeseen valikoitui Nordic nrf52840dk, joka on hyvin monipuolinen ja tehokas kehitysalusta.

5.2.2 Käytetyt sensorit

Käytössä oli kaksi eri MPU6050-sensoria, Olimex MPU6050 ja GY-521 MPU6050. Kumpikin sensoreista sisälsi kiihtyvyyden ja kulmakiihtyvyyden tunnistuksen. Kummankin sensorin kytkentä oli samankaltainen. Testauksen tuloksena käytettiin GY-521-sensoria, sillä Zephyrin sisältämät MPU6050-laiteajurit eivät toimineet Olimex MPU6050:n kanssa. Laiteajureiden tutkiminen ja korjaaminen ei ole todennäköisesti kannattavaa.

Käytössä oli myös ICM-42668- ja BMI270-sensorit. Molemmat sensorit sisälsivät kiihtyvyyden tunnistuksen ja kaltevuuden tunnistuksen. Valitettavasta kumpaakaan sensoria ei saatu toimimaan työn aikana. Syytä ei ehditty selvittää varmaksi, mutta suurin epäilys on laiteajureiden toimimattomuudesta.

Lopulta päädyttiin käyttämään Gy-85 ADXL345-sensoria, joka sisältää ainoastaan kiihtyvyysanturin. Testauksen tuloksena anturi toimi tarvittavalla tavalla ja opinnäytetyötä pääsi edistämään anturin kanssa. Vaikka anturi ei ollut aivan toivotunkaltainen, se kävi myös toimeksiantajalle.

5.3 Toteutus

Toteutus aloitettiin suunnitelman mukaisesti. Testaamalla ensin yksi anturi yhdellä alustalla ja kun tämä toimi yritettiin siirtää samaa toiselle alustalle. Siirron onnistuessa aloitettiin testaamaan anturin vaihtamista toiseen. Ongelmaksi tässä kuitenkin tuli Zephyrin laiteajureiden toimivuus, sillä laiteajurit eivät toimineet kaikilla sensorimalleilla. Zephyr ei ole kaupallinen tuote, joten laiteajureiden tekijät ovat muita Zephyrin käyttäjiä. Tämän vuoksi ajureiden laadusta ja toimivuudesta ei ole varmuutta.

5.3.1 MPU6050-esimerkkisovelluksen testaus ja kehitysalustan vaihto

Toteutus aloitettiin Zephyrin oman esimerkkisovelluksen MPU6050:n pohjalta. Esimerkkisovellus oli kehitetty Nordic nrf52832 -kehitysalustalle, joten sovellus ei suoraan toiminut käytössä olevalla Nordic nrf52840- tai Nucleo F334R8 -alustoilla. Ensimmäisenä sovellus täytyi saada toimimaan kummalla tahansa alustalla ja tähän nrf52840 toimi parempana koekappaleena, sillä se oli hyvin samankaltainen kuin sovelluksen valmiiksi toimiva alusta.

Sovellukseen ensimmäisenä tuli tehdä uusi peittokuva käytetylle alustalle. Kuvassa 6 on peittokuva nrf52840-alustalle. Alustalle on määritelty oletuspinnit tietyille ominaisuuksille, joita voidaan hyödyntää peittokuvan teossa. Kuten kuvassa 5, &i2c0 on hyödynnetty oletuspinnimäärityksiä, jotka nimeävät SDA:n pinniin 26 ja SCL:n pinniin 27. Peittokuvassa alustetaan myös keskeytys pinniin 11.

```
6
7  &i2c0 {
8          mpu6050@68 {
9                  compatible = "invensense,mpu6050";
10                 reg = <0x68>;
11                 status = "okay";
12                 int-gpios = <&gpio0 11 GPIO_ACTIVE_HIGH>;
13         };
14 };
```

Kuva 6. Nrf52840dk_nrf52840.overlay.

Sama ohjelma saadaan myös käännettyä Nucleo f334r8 -alustalle tekemällä sille oma peittokuva, kuva 7. &i2c1 määrittää SDA:n ja SCL:n niitä vastaaviin pinneihin. Keskeytys on määritetty pinniin 6.

```
6
7 &i2c1 {
8         mpu6050@68 {
9                 compatible = "invensense,mpu6050";
10                reg = <0x68>;
11                status = "okay";
12                int-gpios = <&gpiob 6 0 GPIO_ACTIVE_HIGH>;
13        };
14 };
```

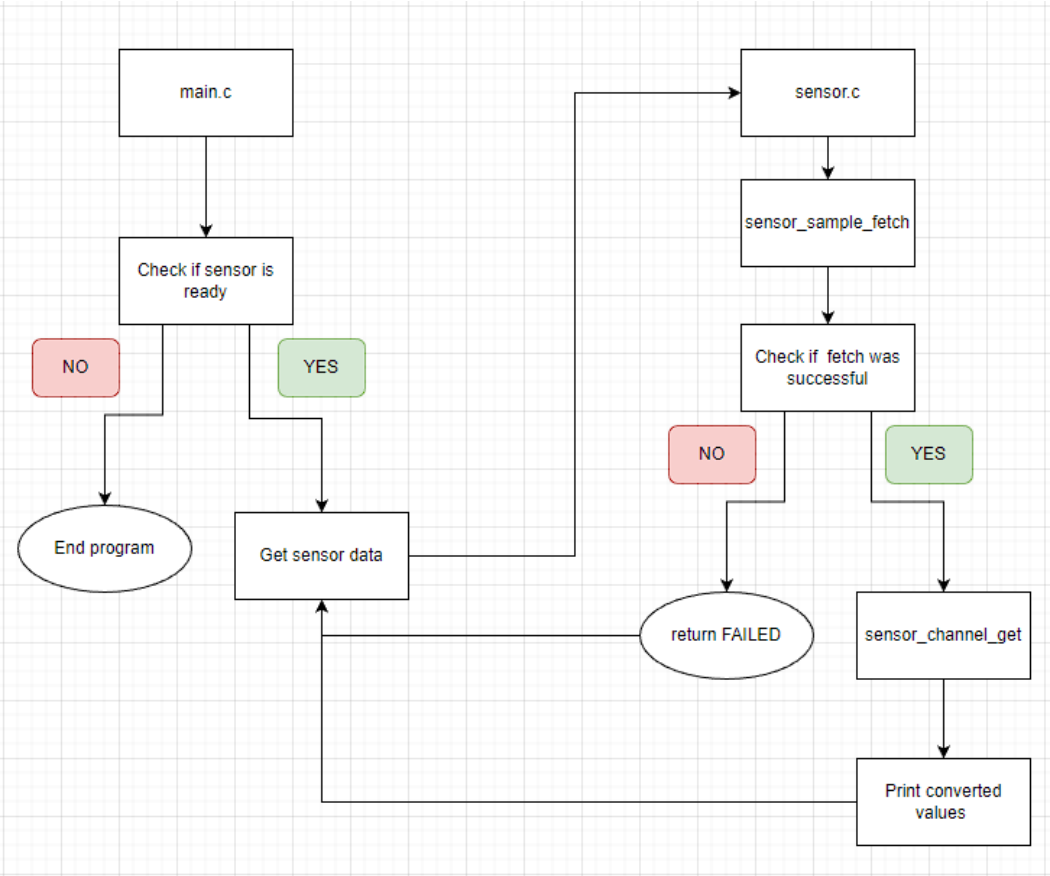
Kuva 7. Nucleo_f334r8.overlay.

Näin esimerkkisovellus on toimiva kahdella eri kehitysalustalla hyvin pienillä muutoksilla. Peittokuva haetaan aina käännöksen yhteydessä, kun käännöskomennossa annetaan kehitysalusta, jolle halutaan ohjelma kääntää. Kääntäjä osaa peittokuvan nimestä katsoa oikean peittokuvan käytettäväksi.

5.3.2 Anturin vaihto

Seuraavaksi aloitettiin testaamaan käytössä olevia antureita. Tässä suureksi esteeksi tuli eri antureiden laiteajureiden toimimattomuus tai yhteensopimattomuus. Laiteajurien ohjelmointi olisi mennyt liian monimutkaiseksi, joten toimivan sensorin löytämiseen meni hyvin kauan. Lopulta toimiva sensori ADXL345 löytyi ja anturin vaihtamista päästiin testaamaan. ADXL345 on kiihtyvyysanturi, joten sen pystyi vaihtamaan MPU6050:n kanssa samaa koodia hyödyntäen.

Sensorin vaihdossa peittokuva piti päivittää halutulle sensorille ja ohjelmaan täytyi myös vaihtaa, minkä nimistä sensoria ohjelma hakee. Sensorin nimen mukana ohjelma osaa hakea oikeita laiteajureita, jotka sisältävät kaikki funktiokutsut. Kuvassa 8 näkyy ohjelman eteneminen. main.c-tiedosto sisältää pääohjelman, joka sensorin löytymisen jälkeen hakee sensorin arvoja ikuisessa luupissa. Sensorin arvot haetaan sensor.c-tiedostossa olevalla funktiolla, minkä jälkeen ne tulostetaan terminaaliin käyttäjän katsottavaksi.



Kuva 8. Vuokaavio ohjelman kulusta

Pääohjelma, kuva 9, sisältää sensorin tietojen hakemisen peittokuvan pinnimäärityksien ja rekisteriosoitteen avulla. Sensorin valmius varmistetaan ensin, jos sensori ei ole valmis ohjelma lopuu. Sensorin ollessa valmis aletaan sensorin arvoja hakea process_sensor-funktiolla ikuisesti.

```c
#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/drivers/sensor.h>
#include <zephyr/drivers/i2c.h>
#include <stdio.h>
#include "sensor.c"

void main(void)
{
    printf("Start sensor test");
    const struct device *const sensor = DEVICE_DT_GET_ONE(adi_adxl345);
    //const struct device *const sensor = DEVICE_DT_GET_ONE(invensense_mpu6050);


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

    printf("Device %p name is %s\n", sensor, sensor->name);

    while (1) {
        int rc = process_sensor(sensor);

        k_sleep(K_SECONDS(2));
    }
}
```

Kuva 9. Pääohjelma main.c

Sensorin arvot haetaan process_sensor-funktiossa, kuvat 10 ja 11. Sensorilta haetaan ensin arvot sensor_sample_fetch-funktiolla, joka palauttaa joko saatujen arvojen määrän tai nollan riippuen käytetystä sensorista. Käytetyn sensorin mukaan tarkistetaan, onnistuiko arvojen haku ja asetetaan adxl-lippu ylös, jos käytetyn anturin nimi on adxl345@53. Lippu auttaa valitsemaan, mitä arvoja sensor_channel_get- funktion tulee muokata luettavaan muotoon. ADXL345-sensori ei anna kulmakiihtyvyys tai lämpötila-arvoja ollenkaan, joten sensor_channel_get-funktio muokkaa ainoastaan kiihtyvyyden arvot.

Muokatut arvot esitellään käyttäjälle lopuksi terminaaliin tulostettuna. Tämän jälkeen ohjelma palaa pääohjelman luuppiin hakemaan arvot uudelleen. Arvoja saadaan, kunnes sensori ei ole enää toiminnasta tai laitteesta katkaistaan virta.

```c
static int process_sensor(const struct device *dev)
{
    struct sensor_value temperature;
    struct sensor_value accel[3];
    struct sensor_value gyro[3];
    int adxl = 0;
    int rc = sensor_sample_fetch(dev);

    if(dev->name == "adxl345@53" && rc != 0)
    {
        adxl = 1;
    }

    if (rc == 0 || adxl == 1) {
        rc = sensor_channel_get(dev, SENSOR_CHAN_ACCEL_XYZ,
                                accel);
    }
    if (rc == 0) {
        rc = sensor_channel_get(dev, SENSOR_CHAN_GYRO_XYZ,
                                gyro);
    }
    if (rc == 0) {
        rc = sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP,
                                &temperature);
    }
```

Kuva 10. Sensorin arvojen hakufunktio sensor.c-tiedostossa

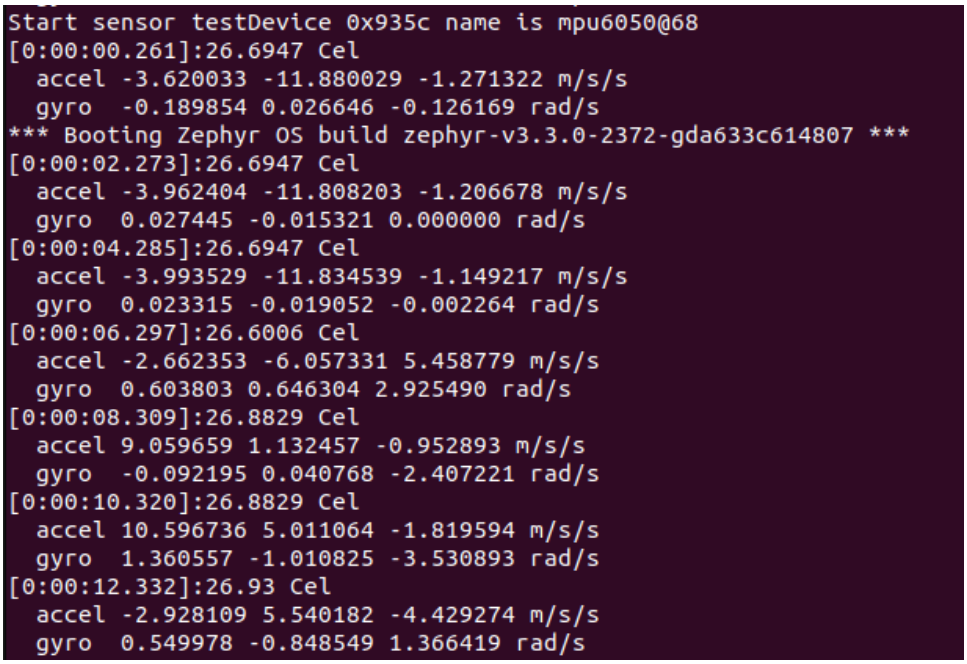
```c
59      if (rc == 0) {
60          printf("[%s]:%g Cel\n"
61                 "  accel %f %f %f m/s/s\n"
62                 "  gyro  %f %f %f rad/s\n",
63                 now_str(),
64                 sensor_value_to_double(&temperature),
65                 sensor_value_to_double(&accel[0]),
66                 sensor_value_to_double(&accel[1]),
67                 sensor_value_to_double(&accel[2]),
68                 sensor_value_to_double(&gyro[0]),
69                 sensor_value_to_double(&gyro[1]),
70                 sensor_value_to_double(&gyro[2]));
71      }
72      if (adxl == 1) {
73          printf("[%s]:"
74                 "  accel %f %f %f m/s/s\n",
75                 now_str(),
76                 sensor_value_to_double(&accel[0]),
77                 sensor_value_to_double(&accel[1]),
78                 sensor_value_to_double(&accel[2]));
79      }
80      else {
81          printf("sample fetch/get failed: %d\n", rc);
82      }
83
84      return rc;
85  }
```

Kuva 11. Sensorin arvojen hakufunktio sensor.c-tiedostossa

5.4 Projektin tulos

Projektin lopputuloksena oli toimiva esimerkkiohjelma, joka täytti kaikki vaatimukset. Ohjelman pystyy siirtämään kehitysalustalta toiselle uuden peittokuvan avulla. Sensoria pystyy vaihtamaan kehitysalustalla muokkaamalla peittokuvan vastaamaan uutta sensoria ja sen kytkentää sekä ohjelmaan vaihtamalla haetun sensorin nimen. Ohjelman ollessa käynnissä se tulostaa anturin arvoja terminaaliin, kuten kuvassa 12 esitetään. Lisää kuvia saaduista tuloksista löytyy liitteestä 1.

Yritys oli hyvin tyytyväinen projektin lopputulokseen. Itse opinnäytetyö toimii ohjeistuksena laiteriippumattoman sovelluksen kehityksessä.



Kuva 12. Kuva terminaalista nrf52840-kehitysalustalla ja mpu6050-sensorilla

5.4.1 Pohdinta

Kehitystyön aikana ohjelmaa ei ehditty saattaa niin laiteriippumattomaksi kuin olisi ollut mahdollista. Sensorien toimimattomuus vei hyvin paljon ohjelman kehitysaikaa ja ohjelma jäi hyvin yksinkertaiseksi. Ohjelma antaa kuitenkin hyvän käsityksen, miten hyvin laiteriippumatonta ohjelmointia voi tehdä ja millaisia ongelmia voi esiintyä.

Kehitystyön aikana opittiin hyvin paljon Zephyr RTOS -käyttöjärjestelmästä sekä laiteriippumattoman sovelluksen kehittämisestä. Työn aikana sensoriongelmia tutkiessa ongelmanratkaisukyky kehittyi.

5.4.2 Kehityskohteet

Ohjelmaa voisi lähteä kehittämään vielä eteenpäin tekemällä sensorin vaihtamisesta vieläkin helpompaa. Sensorin voisi yrittää tunnistaa automaattisesti ja sen tunnistuksen mukaan valita, mitä laiteajureita ohjelma hakee ja käyttää.

Saatuja sensorin arvoja voisi myös parannella erilaisilla kohinanvaimennuksilla ja -suodatuksilla, jotta lopullinen arvo olisi siistimpi. Siistiminen tulisi sijoittaa omaan moduuliinsa ja tehdä mahdollisimman laiteriippumattomaksi, jotta se sopii monelle anturille. Tätä varten antureiden häiriöitä tulisi tutkia enemmän.

6 Lopetus

Opinnäytetyön aikana kehitettiin toimiva esimerkkisovellus ja ohjeistus yrityksen tarpeisiin. Esimerkkisovellus täytti kaikki annetut vaatimukset ja yritys oli hyvin tyytyväinen lopputulokseen. Sovellus oli helposti siirrettävissä kehitysalustalta toiselle ja kehitysalustan sensoria pystyi pienillä koodi muutoksilla ja peittokuvan muokkaamisella vaihtamaan toiseen sensoriin. Itse opinnäytetyö toimii vaadittuna ohjeistuksena.

Esimerkkisovelluksen teko olisi pitänyt aloittaa aiemmin, jotta sen viimeistelyyn ja paranteluun olisi jäänyt aikaa lopussa. Sovelluksen kehitys jäi vähän kesken ja sovellusta olisi voinut parantaa esimerkiksi signaalin kohinanvaimennuksilla.

Työn esimerkkisovelluksen luomisessa oli hyvin paljon ongelmia sensoreiden toimivuuden kanssa, mutta ongelmien tutkiminen ja ratkaiseminen antoivat paljon oppia niin Zephyr-käyttöjärjestelmän toiminnasta kuin laiteajureiden tutkimisesta. Myös lopullinen dokumentointivaihe oli hyvin haastava ja raskas toteuttaa, sillä asiatekstin kirjoittaminen vaati hyvin paljon aikaa.

Opinnäytetyön perusteella voi todeta, että laiteriippumattoman ohjelman kehittäminen on järkevää ja toimivaa sulautetuilla järjestelmillä. Laiteriippumattoman ohjelman tekemisen vaikeudet määräytyvät paljon käyttöjärjestelmän mukaan. Zephyr RTOS -käyttöjärjestelmä tulee laiteriippumattomuutta todella hyvin ja sillä on helppoa kehittää laiteriippumaton ohjelma.

Kokonaisuudessaan työn tekeminen oli hyvin mielenkiintoista ja opettavaista. Uskon, että työn antamista opeista on tulevaisuudessa paljon hyötyä työelämässä.

Lähteet

1. Käyttöjärjestelmän tehtäviä. MOOC.fi [internet]. 2022. [viitattu 10.4.2023] Saatavilla: https://tito-perusteet-2022.mooc.fi/luku-4/1-kayttojarjestelman-tehtavat

2. Cross-platform-development. Technopedia [internet]. [viitattu 2.3.2023]. Saatavilla: https://www.techopedia.com/definition/30026/cross-platform-development

3. Ahopelto M. Device-independent code with model-based design. Devecto [internet] 2019. [viitattu 27.4.2023] Saatavilla: https://devecto.com/en/kohti-laitteistoriippuma-tonta-koodia-mallipohjaisella-suunnittelulla/

4. Johnson M. Modular Embedded Application Architecture Can Reduce Costs and Increase Reliability. DORNERWORKS [internet] 2019. [viitattu 16.3.2023] Saatavilla: https://dornerworks.com/blog/modular-embedded-software/

5. Guide to Modular Firmware. Hackster.io [internet]. 2019. [viitattu 16.3.2023] Saatavilla: https://www.hackster.io/LuckyResistor/guide-to-modular-firmware-6c5409

6. The advantages of Modular Software and Programming. Gwentech embedded [internet]. [viitattu 16.3.2023]. Saatavilla: http://gwentechembedded.com/the-advantages-of-modular-software-and-programming/

7. Meaning of Device independeence. ENCYCLO.CO.UK [internet]. 2023 [viitattu 23.3.2023]. Saatavilla: https://www.encyclo.co.uk/meaning-of-Device_independence

8. Zehyr overview. Zephyr RTOS [internet]. [viitattu 15.3.2023]. Saatavilla: https://www.zephyrproject.org/wp-content/uploads/sites/38/2023/01/Zephyr-Overview-20230124.pdf

9. Getting started guide. Zephyr RTOS. [viitattu 15.3.2023] Saatavilla: https://docs.zephyrproject.org/latest/develop/getting_started/index.html

10. Application Development. Zephyr RTOS. [viitattu 15.5.2023] Saatavilla: https://docs.zephyrproject.org/latest/develop/application/index.html

11. Devicetree. Zephyr RTOS. [viitattu 15.3.2023]. Saatavilla: https://docs.zephyrproject.org/latest/build/dts/index.html#devicetree

12. STM32 Nucleo-64 development board with STM32F334R8 MCU, supports Arduino and ST morpho connectivity. [viitattu 25.4.2023] Saatavilla: https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-mpu-eval-tools/stm32-mcu-mpu-eval-tools/stm32-nucleo-boards/nucleo-f334r8.html
13. About Nordic Semiconductor. Nordic Semiconductor. [viitattu 25.4.2023] Saatavilla: https://www.nordicsemi.com/About-us

NucleoF334R8 MPU6050

```
Start sensor testDevice 0x80079a8 name is mpu6050@68
[0:00:00.005]:33.5182 Cel
  accel 0.000000 0.000000 0.000000 m/s/s
  gyro  0.000000 0.000000 0.000000 rad/s
*** Booting Zephyr OS build zephyr-v3.3.0-2372-gda633c614807 ***
[0:00:02.015]:30.0359 Cel
  accel 5.339069 5.295974 0.196324 m/s/s
  gyro  0.005995 -0.029710 -0.046764 rad/s
[0:00:04.026]:30.13 Cel
  accel 2.913743 2.176329 0.711077 m/s/s
  gyro  -4.365720 3.444693 3.727942 rad/s
[0:00:06.036]:30.0829 Cel
  accel 2.403778 14.982913 6.885723 m/s/s
  gyro  0.589548 0.622589 -0.118176 rad/s
[0:00:08.046]:30.13 Cel
  accel -6.356606 3.869029 -7.659052 m/s/s
  gyro  -0.258202 -3.361023 -1.107551 rad/s
[0:00:10.056]:30.0359 Cel
  accel -10.153810 -10.946290 3.241749 m/s/s
  gyro  -3.422576 -4.365720 2.928022 rad/s
[0:00:12.067]:29.9418 Cel
  accel 3.718195 6.411672 -1.793258 m/s/s
  gyro  0.011591 0.044499 -0.072477 rad/s
[0:00:14.077]:29.9888 Cel
  accel 3.869029 6.177040 -1.984794 m/s/s
  gyro  -0.010924 -0.035439 -0.103787 rad/s
```

NucleoF334R8 ADXL345

```
Start sensor testDevice 0x8007340 name is adxl345@53
[0:00:00.009]:  accel 9.500192 -1.532289 -3.064578 m/s/s
*** Booting Zephyr OS build zephyr-v3.3.0-2372-gda633c614807 ***
[0:00:02.022]:  accel 9.500192 -1.838746 -3.064578 m/s/s
[0:00:04.034]:  accel 9.193734 -1.838746 -3.064578 m/s/s
[0:00:06.047]:  accel -12.564770 5.209782 0.919373 m/s/s
[0:00:08.059]:  accel 3.677493 2.758120 -2.145204 m/s/s
[0:00:10.071]:  accel 7.048529 -4.290409 -3.064578 m/s/s
[0:00:12.084]:  accel 3.371035 11.032481 5.822698 m/s/s
[0:00:14.096]:  accel 11.951854 -1.225831 -2.451662 m/s/s
[0:00:16.109]:  accel 9.193734 -1.838746 -3.983951 m/s/s
[0:00:18.121]:  accel 9.193734 -1.532289 -3.983951 m/s/s
[0:00:20.133]:  accel 4.596867 -9.500192 -17.468095 m/s/s
[0:00:22.146]:  accel 40.145973 6.129156 -5.516240 m/s/s
[0:00:24.158]:  accel -1.838746 -1.532289 -12.258312 m/s/s
[0:00:26.171]:  accel 4.596867 2.451662 -5.822698 m/s/s
[0:00:28.183]:  accel 9.500192 0.000000 -3.371035 m/s/s
```

Nordic nrf52840 ADXL345