VAIHTOEHTOISTEN OHJELMOINTIKIELTEN HYÖDYNTÄMINEN SULAUTETUISSA JÄRJESTELMISSÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, Insinööri (AMK)

2023

Riku Sjöroos

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka Tiivistelmä
Tekijä Riku Sjöroos Vuosi 2023
Työn nimi Vaihtoehtoisten ohjelmointikielten hyödyntäminen sulautetuissa järjestelmissä
Ohjaaja Timo Karppinen

Sulautettujen järjestelmien kehityksessä on pitkään ollut hallitsevana ohjelmointikielenä C-ohjelmointikieli. Nykyään kuitenkin on tarjolla myös vaihtoehtoisia ohjelmointikieliä C-ohjelmointikielen tilalle.

Tämän opinnäytetyön tavoitteena oli selvittää, mitä muita ohjelmointikieliä olisi tarjolla sulautettujen järjestelmien kehitykseen ja testata niillä yleisimpiä mikrokontrollereiden ominaisuuksia. Työn tilaajana oli Huld Oy.

Ohjelmointikieliksi valikoitui kartoituksen jälkeen Python ja JavaScript. Python-implementaationa käytettiin MicroPythonia ja JavaScript-implementaationa Espruinoa. Verrokkina näille kahdelle käytettiin Arduino-ohjelmointikieltä, joka perustuu C++-ohjelmointikieleen. Työssä kartoitettiin yleisimpien valmistajien mikrokontrollereita ja niistä pyrittiin valitsemaan sellainen, jota valitut ohjelmointikielet tukevat. Kehitysalustaksi valikoitui kartoituksen jälkeen STM32F4DISCOVERY-kehitysalusta, jolla testattiin kaikkia kolmea eri ohjelmointikieltä.

Ohjelmointikielillä testattiin suorituskykyä laskennassa, sekä muutamaa yleistä ominaisuutta kuten GPIO-pinnejä, A/D-muunninta, D/A-muunninta, SPI-väylää, pulssinleveysmodulaatiota ja toimintojen ajoittamista.

Laskennan ja koodin suorituskyvyssä C++-pohjainen Arduino-ohjelmointikieli oli ylivoimainen. Toiseksi nopein oli MicroPython ja hitain Espruino. Kaikilla ohjelmointiympäristöillä pystyi suorittamaan testattavat asiat ongelmitta.

MicroPython ja Espruino mielletään enemmän opetuskäyttöön ja harrastajille sopiviksi, mutta ne soveltuisivat myös erinomaisesti nopeaan kehitykseen ja prototyyppien kehittämiseen. Opetuskäytössä ne ovat erinomaisia, sillä ne tarjoavat REPL-komentokehotteen, johon syötetyt komennot suoritetaan mikrokontrollerissa välittömästi. Ohjelmakoodia ei myöskään tarvitse kääntää, kuten C-pohjaisilla ohjelmointikielillä. Tämä nopeuttaa pienten muutosten tekemistä ohjelmakoodin ja sen nopeaa testaamista.

Avainsanat MicroPython, Espruino, mikrokontrolleri
Sivut 41 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Author Riku Sjöroos
Subject Utilizing Alternative Programming Languages in Embedded Systems
Supervisor Timo Karppinen

Abstract
Year 2023

For a long time, the C programming language has been the most popular programming language in embedded systems. Today there are more alternative languages from which you can choose a programming language for an embedded system.

The goal of this thesis was to research what alternative languages there are for embedded systems and test them with basic microcontroller features. The commissioner of this thesis was Huld Oy.

After conducting research Python and JavaScript were chosen. The Python implementation for embedded systems was MicroPython and the JavaScript implementation was Espruino. A third programming language was Arduino-programming language which is based on the C++ programming language, and it was used as control language in this thesis. A microcontroller for this work was selected among common microcontrollers. The chosen evaluation board was STM32F4DISCOVERY evaluation board which was supported by all three programming environments.

The programming languages were tested with a calculation task and with some basic functionaltiies of a microcontroller like GPIO-pins, ADC-converter, DAC-converter, SPI-bus, PWM and timing of events.

In the calculation task and execution speed C++ based Arduino was the fastest. The second was MicroPython and the slowest one was Espruino. All three programming environments supported the tested features without problems.

MicroPython and Espruino are often thought of as educational and meant for hobbyist, but those can also be used for prototyping. For educational purposes, these are great because of the REPL-prompt which can be used to send commands to microcontroller in real time. Program code does not require compilation, making it much faster and easier to make minor changes to program code and test them immediately.

Keywords MicroPython, Espruino, microcontroller
Pages 41 pages

Sisällys

Kuvat, taulukot ja kaavat

1 Johdanto

Pitkään käytössä ollut ja edelleenkin suosituin ohjelmointikieli sulautettujen järjestelmien kehityksessä on C-ohjelmointikieli, joka on kehitetty jo 1970-luvulla. Tämän opinnäytetyön tarkoituksena olikin vertailla muita sulautettujen järjestelmien kehittämiseen soveltuvia ohjelmointikieliä, ja arviodia niiden soveltuvuutta erilaisiin käyttötarkoituksiin, kuten prototyyppien kehitykseen.

Opinnäytetyöhön valittiin vertailukieleksi C++-pohjainen Arduino-ohjelmointikieli, jonka lisäksi valittiin kaksi vaihtoehtoista ohjelmointikieltä. Arduino-ohjelmointiympäristö on suosittu ympäristö sulautetuissa järjestelmissä varsinkin harrastajien keskuudessa, mutta se tarjoaa myös nykyään enemmän ammattikäyttöön soveltuvaa tekniikkaa. Muut kaksi ohjelmointikieltä valittiin tutkimalla verkosta eri vaihtoehtoja.

Ohjelmointikieliä testattiin samalla kehitysalustalla, jotta mahdollisia eroja suorituskyvyssä saataisiin selville. Kielillä testattiin laitteiston ominaisuuksia ja selvitettiin tukevatko ne yleisimpiä tekniikoita. Testaamisen jälkeen ohjelmointikieliä verrattiin keskenään ja tehtiin päätelmiä niiden soveltuvuudesta. Tämän opinnäytetyön tilaaja oli Huld Oy.

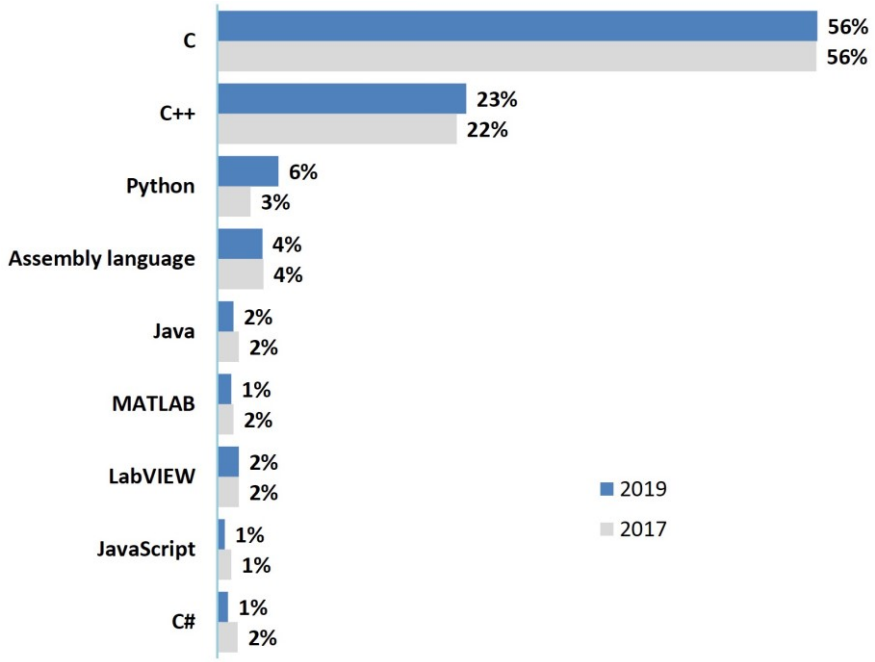
2 Suunnittelu

Työn suunnitteluvaihe jakautui kolmeen eri vaiheeseen. Ensimmäisessä vaiheessa selvitettiin, mitä eri vaihtoehtoisia ohjelmointikieliä on saatavilla mikrokontrollereille. Kartoituksen jälkeen valittiin kolme vertailtavaa kieltä. Ohjelmointikielien valinnan jälkeen selvitettiin, mitkä eri mikrokontrollerit kyseisiä ohjelmointikieliä tukevat ja niistä valittiin yksi, jolla vertailu suoritettiin. Mikrokontrollerin ja ohjelmointikielien valinnan jälkeen selvitettiin sopiva laskutehtävä suoritettavaksi eri ohjelmointikielillä. Laskutehtävän lisäksi valittiin muutamia mikrokontrollerin perusominaisuuksia, joita testattiin.

2.1 Ohjelmointikielten valinta

Sulautetuissa järjestelmissä on pitkään ollut suosituimpana kielenä C-ohjelmointikieli, mutta nykyään myös muiden kielten suosio on noussut. Todennäköisesti Python-ohjelmointikielen suosio tulee nousemaan entisestään tekoälyn yleistyessä (Evanczuk, 2019). Kuvassa yksi on esitetty eri ohjelmointikielten suosiota sulautetuissa järjestelmissä toisiinsa verrattuna.

Kuva 1. Ohjelmointikielien suosio sulautetuissa järjestelmissä (Evanczuk, 2019).



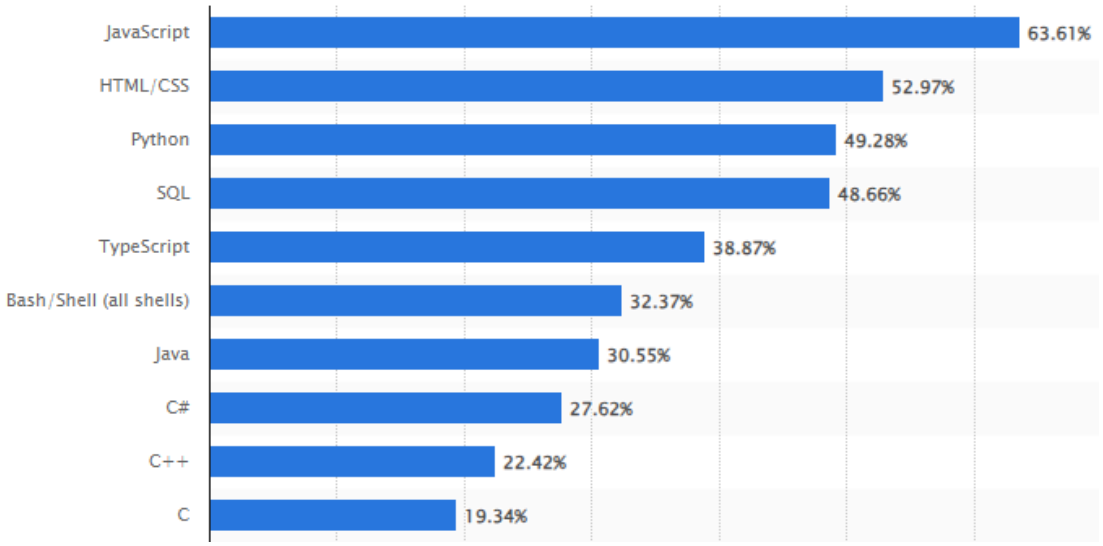
Kuvasta voidaan huomata, että Pythonin osuus on ollut kasvussa ja tästä syystä opinnäytetyön yhdeksi vertailtavaksi kieleksi valittiin MicroPython, joka on Pythonista kehitetty mikrokontrollereille optimoitu avoimen lähdekoodin versio (MicroPython, n.d.).

Python on vuonna 1991 julkaistu korkean tason yleiskäyttöinen ohjelmointikieli. Sen kehitti alun perin Guido Van Rossum. Pythonin kehityksessä painotettiin erityisesti koodin luettavuutta. (Paramanick, n.d.) Pythonia käytetään muun muassa web-ohjelmoinnissa, data-analyyseissa, käyttöliittymissä ja opetuksessa (Python, n.d.). Pythonia pidetään hyvänä ohjelmointikielenä ohjelmoinnin opettelussa, koska sen syntaksi on helppolukuista. Python on niin sanottu tulkattava kieli, eli sitä ei tarvitse kääntää vaan tulkin avulla voidaan tulkata

kieltä heti, kun se on kirjoitettu. Yleisesti ottaen tulkattavien kielten suorituskyky on alhaisempi kuin käännettävien kielten. (W3Schools, n.d.)

Toiseksi ohjelmointikieleksi valittiin JavaScript, joka on vielä varsin harvinainen sulautetuissa järjestelmissä. JavaScript on erityisesti web-ohjelmoinnissa käytetty kieli ja se on tällä hetkellä käytetyin ohjelmointikeli (Kuva 2).

Kuva 2. Ohjelmointikielien suosio (Statista, 2023).



JavaScriptin kehitti vuonna 1995 Brendan Eich. JavaScriptiä käytetään web-ohjelmoinnissa HTML:n ja CSS:n rinnalla. JavaScriptillä tuotetaan sivuille toiminnallisuus ja käyttäytyminen, joka mahdollistaa käyttäjän vuorovaikutuksen sivun kanssa. Ennen kuin JavaScript otettiin käyttöön, sivut olivat staattisia, eivätkä ne mahdollistaneet käyttäjän vuorovaikutusta. JavaScript on Pythonin tavoin tulkattava kieli. (Dickson, 2022) Tässä työssä käytetään JavaScriptiä Espruino-implementaation avulla, joka on mikrokontrollereille suunniteltu avoimen lähdekoodin tulkki.

Kolmanneksi kieleksi valittiin C++-ohjelmointikieli. C++ pohjautuu C-ohjelmointikieleen, joka on edelleen käytetyin ohjelmointikieli sulautetuissa järjestelmissä, kuten aikaisemmin esitetystä kuvasta voidaan todeta (Kuva 1). C++-ohjelmointikieli on pitkään käytössä ollut kieli, sillä se kehitettiin jo vuona 1979 Bjarne Stroustrupin toimesta. C++ on yleiskäyttöinen keskitason ohjelmointikieli, joka tarjoaa pääsyn matalan ja korkean tason muistinkäsittelyyn.

C ja C++ ovat käännettäviä kieliä, eli ohjelma pitää kääntää kääntäjäohjelmalla ennen kuin ohjelmaa voidaan suorittaa. (Kumari, n.d.) Tässä työssä käytettiin Arduino-ohjelmointikieltä, joka perustuu C++-ohjelmointikieleen.

2.2 Mikrokontrollerin valinta

Mikrokontrollereiden kartoittaminen aloitettiin tutkimalla internetistä eri vaihtoehtoja, jotka tukevat valittuja ohjelmointiympäristöjä. Mahdolliset mikrokontrollerit taulukoitiin vertailua helpottamaan. Vertailuun otettiin mukaan yleisesti tunnettujen valmistajien kehitysalustoja, joissa oli eri mikrokontrollerit. Alla olevassa taulukossa on esitetty vertaillut kehitysalustat.

Taulukko 1. Kehitysalustojen vertailu.

Kehitysalusta	Espruino	MicroPython	C/C++
Raspberry Pi Pico	✗	✓	✓
ESP32	✓	✓	✓
STM32F4DISCOVERY	✓	✓	✓
Arduino Nano	✗	✓	✓
nRF52 DK	✓	✓	✓

Taulukosta huomataan, että monelle eri kehitysalustalle on vaihtoehtoisia ohjelmointiympäristöjä saatavilla. Vertailussa oli mukana vain murto-osa alustoista, koska niitä on saatavilla valtava määrä ja isomman määrän vertailu ei olisi ollut järkevää. Raspberry Pi Pico:lle on myös saatavilla JavaScript implementaatio Kaluma, mutta se on tarkoitettu ainoastaan Raspberry Pi Pico:lle, joten sitä ei voida pitää niin yleiskäyttöisenä.

Testattavaksi alustaksi valittiin STM32F4DISCOVERY (Kuva 3). Se on STMicroelectronics:n valmistama kehitysalusta, jossa on 168 megahertsin STM32F407VGT6-mikrokontrolleri 32 bittisellä Arm Cortex M4 ytimellä. Kehitysalustassa on megatavun flash-muisti ja 192 kilotavua RAM-muistia. Kehitysalustalla on myös muita lukuisia ominaisuuksia, jotka helpottavat nopeaa testailua, kuten LED-valoja, kiihtyvyysanturi, mikrofoni ja painonappeja. (STMicroelectronics, n.d.a)

Kuva 3. STM32F4DISCOVERY kehitysalusta (STMicroelectronics, n.d.a).

2.3 Laskentatehtävän valinta

Testattavien ohjelmointikielten ja kehitysalustan valitsemisen jälkeen valittiin laskentatehtävä ohjelmointikielten suorituskyvyn erojen selvittämiseksi. Sopivan laskutehtävän selvittäminen aloitettiin tutkimalla internetistä sopivia laskutehtäviä. Yleisesti käytetty laskutehtävä on piin likiarvon laskeminen. Piin likiarvon laskemiseen voidaan käyttää Leibnizin kaavaa (Kaava 1).

Kaava 1. Leibnizin kaava piin likiarvon laskemiseksi.

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

Leibnizin kaavalla saadaan sitä tarkempi piin likiarvo, mitä pidemmälle laskemista jatketaan. Tämän vuoksi kaava soveltuu hyvin testaamiseen, koska sillä saadaan ohjelma suorittamaan useita iteraatiokierroksia.

2.4 Mikrokontrollerin ominaisuuksien hyödyntäminen ja ajoituksen testaaminen

Laskentakyvyn lisäksi valittiin muutama testattava ominaisuus kehitysalustalla. Kehitysalustalla päätettiin testata GPIO-pinnejä, A/D-muunninta, SPI-tiedonsiirtoa, pulssinleveysmodulaatiota ja D/A-muunninta. Näiden lisäksi on myös monia muita ominaisuuksia, mutta ne on rajattu tämän työn ulkopuolelle. Ominaisuuksien lisäksi testattiin miten, eri toimintoja voidaan ajastaa mikrokontrollerilla eri ohjelmointiympäristöissä.

GPIO-pinnillä (General Purpose Input Output) tarkoitetaan yleiskäyttöistä sisääntulo- tai ulostulopinniä. Eli ohjelmassa voidaan määrittää pinni, joko ulostuloksi tai sisääntuloksi. Ulostulopinnin tila voidaan määrittää joko todeksi tai epätodeksi. Mikäli pinni määritetään ulostuloksi ja sen arvo asetetaan todeksi syöttää mikrokontrolleri pinniin jännitteen, jos taas pinni on määritetty sisääntuloksi, voidaan tarkkailla, johdetaanko pinniin jännitettä. (Nerdy Electronics, n.d.)

A/D-muuntimella (Analog to Digital Converter) tarkoitetaan muunninta, joka muuttaa analogisen signaalin digitaaliseksi. A/D-muuntimella voidaan esimerkiksi havainnoida jännitteen voimakkuutta. (Kirvan, n.d.) D/A-muunnin (Digital to Analogue Converter) on laite joka, muuttaa binääridatan analogiseksi signaaliksi (University of Plymouth, n.d.).

SPI (Serial Peripheral Interface) on sarjaväylä, jolla voidaan siirtää dataa kahden laitteen välillä. SPI hyödyntää isäntälaitteen lähettämää tahdistussignaalia, jonka mukaan laitteet kommunikoivat. (Sparkfun, n.d.)

Pulssinleveysmodulaatiolla tarkoitetaan tekniikkaa, jolla voidaan muuttaa ulostulon lähtöjännite pulssiseksi eli lähdön arvoa toden ja epätoden välillä muutetaan. Pulssin taajuutta ja pituutta voidaan muuttaa ohjelmassa.

3 Toteutus

Työn toteutus aloitettiin asentamalla tarvittavat ohjelmat tietokoneelle ja tutustumalla kunkin ohjelmointiympäristön käyttöön saattamiseen. Asennusten jälkeen suoritettiin kappaleissa 2.3 ja 2.4 mainitut testit jokaisella ympäristöllä vuorotellen.

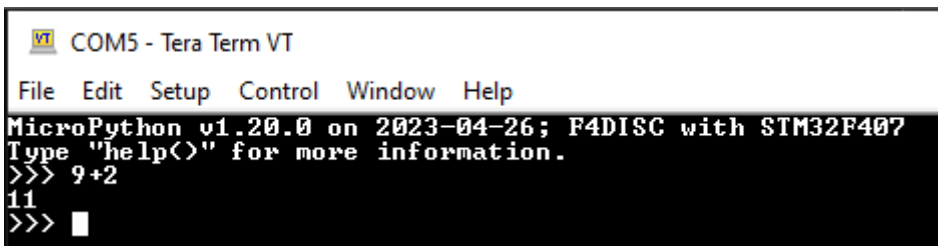
3.1 MicroPythonin testaaminen

3.1.1 Käyttöönotto

MicroPythonin asentaminen mikrokontrollerilla aloitettiin tutustumalla MicroPythonin dokumentaatioon. Dokumentaatiosta selvisi, että eri kehitysalustoille on oma versio MicroPythonista, jonka voi ladata ilmaiseksi MicroPythonin verkkosivuilta hex-tiedostona. Lataamisen jälkeen kehitysalusta piti ohjelmoida tiedostoa käyttäen, jotta tulkki saataisiin kehitysalustalle. Tässä työssä käytetyn STMicroelectronics:n kehitysalustan ohjelmointi onnistui STM32CubeProgrammer-ohjelmalla, jonka voi ladata ilmaiseksi verkosta.

Ennen ohjelmointia kehitysalusta kytkettiin tietokoneeseen USB-johdolla. STM32CubeProgrammer-ohjelmasta valittiin hex-tiedosto ja valittiin ”Start Programming”. Ohjelmoinnin jälkeen kehitysalustan USB-johto irrotettiin ja kytkettiin takaisin tietokoneeseen. Tämän jälkeen tietokone tunnisti uuden levyaseman nimeltä ”PYBFLASH”. Asennuksen onnistuminen voitiin myös todeta ottamalla yhteys MicroPythonin REPL-komentokehotteeseen, joka on siis interaktiivinen Python tulkki. Yhteyden voi muodostaa millä tahansa terminaaliohjelmalla, joka pystyy sarjaporttiliikenteeseen. Yhteyden muodostaminen onnistui ja tulkkia testattiin yksinkertaisella laskutoimituksella (Kuva 4).

Kuva 4. MicroPython REPL-komentokehote.



3.1.2 Oman ohjelman ohjelmointi alustalle

PYBFLASH-asemalle siirrettiin tekstieditorissa muokattu Python-tiedosto, jonka jälkeen MicroPython alkoi suorittamaan ohjelmaa, kun se käynnistettiin uudelleen. Ohjelman tulee olla oletuksena nimetty nimellä ”main”, jotta se suoritetaan. Esimerkkinä vilkutetaan LED-valoa viisi kertaa, joka on kytketty pinniin PA1 ja samalla luetaan pinnin PA5 arvoa, joka on kytketty LED-valoon (Kuva 5). Samalla voidaan tarkkailla REPL-komentokehotteesta tulostettua sisääntulon arvoa. (Kuva 6).

Kuva 5. GPIO esimerkki MicroPythonilla.

```
# Tarvittavat tuonnit
from machine import Pin
from time import sleep

# Pinnin asettaminen ulostuloksi
led = Pin("PA1", Pin.OUT)

# Pinnin asettaminen sisääntuloksi
input = Pin("PA5", Pin.IN)

for i in range(10):
    if i % 2 == 0:
        led.value(1)
        print("Sisaantulon arvo " + str(input.value()))
    else:
        led.value(0)
        print("Sisaantulon arvo " + str(input.value()))
    sleep(1)
```

Kuva 6. Sisääntulon arvo komentokehotteesta nähtynä.

```
MicroPython v1.20.0 on 2023-04-26; F4DISC with STM32F407
Type "help()" for more information.
>>> Sisaantulon arvo 1
Sisaantulon arvo 0
Sisaantulon arvo 1
Sisaantulon arvo 0
Sisaantulon arvo 1
Sisaantulon arvo 0
Sisaantulon arvo 1
Sisaantulon arvo 0
Sisaantulon arvo 1
Sisaantulon arvo 0
```

3.1.3 Laskentanopeuden testaaminen

Suorituskyvyn testaamiseen käytettiin Leibnizin kaavaa (Kaava 1) piin likiarvon laskemiseen. Kaava muutettiin Python-ohjelmaksi, ja lisäksi ohjelmaan lisättiin toiminnallisuus kuluneen ajan mittaamiseen (Kuva 7). Laskennan tulokset voidaan tarkistaa REPL-komentokehotteesta (Kuva 8).

Kuva 7. Piin likiarvon laskeminen Leibnizin kaavalla ja kuluneen ajan mittaaminen.

```
# Tarvittavat tuonnit
import time

#Laskee piin käyttämällä Leibnizin kaavaa
def laske_pii(n):
    pii = 0
    osoittaja = 1
    for i in range(n):
        if i % 2 == 0:
            pii += 4 / osoittaja
            osoittaja += 2
        else:
            pii -= 4 / osoittaja
            osoittaja += 2
    return(pii)

print("Aloitetaan piin laskeminen...")
time.sleep(1)
aloitus = time.ticks_ms()
pii = laske_pii(1000000)
# Laskee kuluneen ajan
erotus = time.ticks_diff(time.ticks_ms(), aloitus)
time.sleep(1)
print("Piin laskemiseen kului aikaa " + str(erotus) + " millisekuntia")
print("Piin arvo " + str(pii))
```

Kuva 8. Laskennan tulokset.

```
>>> MicroPython v1.20.0 on 2023-04-26; F4DISC with STM32F407
Type "help()" for more information.
>>> Aloitetaan piin laskeminen...
Piin laskemiseen kului aikaa 14538 millisekuntia
Piin arvo 3.141595
```

Laskennassa käytettiin miljoonaa iteraatiokierrosta ja aikaa laskemiseen kului 14538 millisekuntia. Arvossa on viisi ensimmäistä desimaalia pilkun jälkeen oikein, joten melko hyvään tarkkuuteen tällä iteraatiomäärällä päästiin. Laskenta suoritettiin viisi kertaa ja tulokset taulukoitiin.

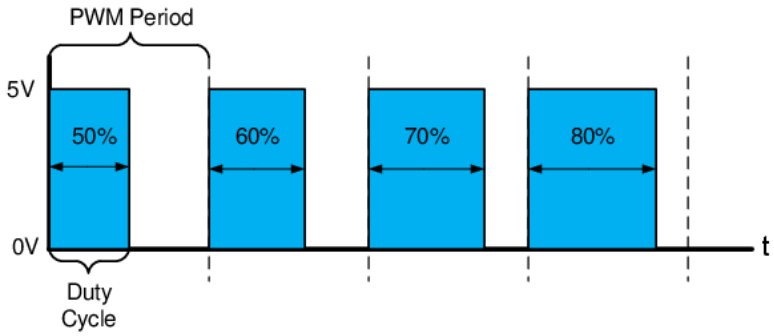
3.1.4 Pulssinleveysmodulaation testaaminen

Pulssinleveysmodulaatiota testattiin LED-valon avulla. Pulssinleveysmodulaatio onkin yleisesti käytetty tekniikka LED-valojen himmennyksessä. LED-valo kytkettiin pulssinleveysmodulaatiota tukevaan pinniin PA1, ja tämän jälkeen kirjoitettiin ohjelmakoodi, joka muutti LED-valon kirkkautta (Kuva 9). Pulssinleveyden lisäksi MicroPythonilla voidaan säätää pulssin taajuutta. Kuvasta 10 voidaan päätellä, että mikäli pulssin pituus on pidempi, palaa LED-valo kirkkaammin, kunhan taajuus on riittävän suuri. Mikäli taajuus on liian pieni, havaitaan silmällä LED-valon vilkkumista.

Kuva 9. Pulssinleveysmodulaation testaaminen.

```
# Vaadittavat tuonnit
from pyb import Pin, Timer, delay

# Pinnin valinta
pinni = Pin("PA1")
# Ajastimen valinta, Timer 2, taajuus 1000 Hz
ajastin = Timer(2, freq=1000)
# Valitaan kanava 2, PWM moodi, ja haluttu pinni
kanava = ajastin.channel(2, Timer.PWM, pin=pinni)
i = 1

# Muutetaan pulssinleveyttä 1-100%
while(1):
    if i < 100:
        i = i + 1
    else:
        i = 1
    kanava.pulse_width_percent(i)
    delay(10)
```

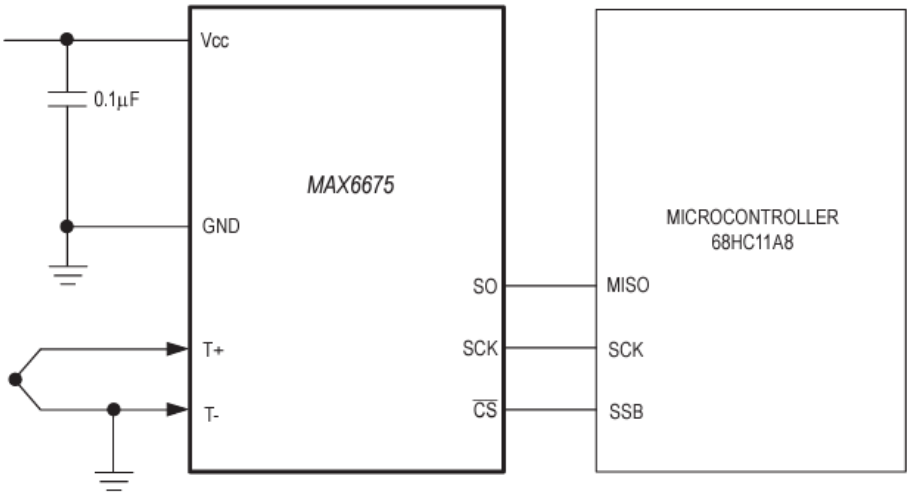
Kuva 10. Pulssinleveysmodulaatio (Microchip, n.d.).



3.1.5 Serial Peripheral Interface (SPI):n lukeminen

SPI-väylän lukemisen testaaminen aloitettiin kytkemällä MAX6675-termoparimuunnin kehitysalustaan MAX667-dokumentaation ohjeiden mukaisesti (Kuva 11). Muuntimeen oli kytketty K-tyypin termopari. Termoparilla luettiin huoneen lämpötilaa.

Kuva 11. MAX6675-kytkentä (Analog Devices, n.d.).



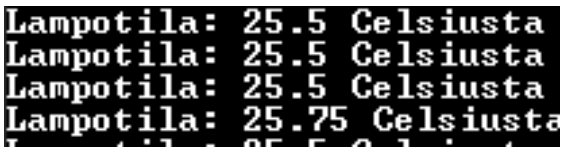
Muuntimen SO (Serial Data Output) kytkettiin kehitysalustan SPI1-väylän MISO (Master In Slave Out) pinniin, SCK (Serial Clock) kytkettiin SPI1-väylän SCK pinniin ja CS (Chip Select) kytkettiin pinniin PA1. Jännite ja maa kytkettiin kehitysalustan vastaaviin pinneihin. Tämän jälkeen kirjoitettiin Pythonilla ohjelmakoodi mikrokontrollerille (Kuva 12).

Kuva 12. Lämpötilan lukeminen SPI-väylää käyttäen.

```
# Tarvittavat tuonnit
from machine import SPI, Pin
import time

# Määritetään SPI1-väylä
spi = SPI(1, baudrate=1000000, polarity=0, phase=0)
# Määritetään CS (Chip Select) pinni
cs = Pin("PA1", Pin.OUT)

# Luetaan lämpötila
def lue_lampotila():
    cs.off()
    time.sleep_ms(20)
    # Luetaan data SPI väylästä
    data = spi.read(2)
    cs.on()
    # Muunnetaan luetut tiedot lämpötilaksi
    arvo = (data[0] << 8) | data[1]
    lampotila = (arvo >> 3)  * 0.25
    return lampotila

while True:
    lampotila = lue_lampotila()
    print("Lampotila: "+ str(lampotila) + " Celsiusta")
    # Odotetaan kaksi sekuntia ennen seuraavaa lukemaa
    time.sleep(2)
```

Ohjelman suorittamisen jälkeen huomattiin REPL-komentokehotteesta, että lämpötilan lukeminen onnistui (Kuva 13).

Kuva 13. Lämpötilan lukeminen REPL-komentokehotteesta.



3.1.6 A/D-muuntimen testaaminen

A/D-muunninta testattiin kytkemällä säätövastus eli potentiometri kehitysalustaan. Potentiometrin ulostulopinni kytkettiin kehitysalustan pinniin PA1. Tämän jälkeen kirjoitettiin ohjelmakoodi ulostulon arvon lukemiseksi (Kuva 14).

Kuva 14. A/D-muuntimen arvon lukeminen MicroPythonilla.

```
# Vaadittavat tuonnit
from machine import ADC
import time

# Valitaan ADC pinni
adc = ADC("PA1")

while(True):
    # Luetaan arvo (0-65535) ja tulostetaan se
    arvo = adc.read_u16()
    print(arvo)
    # Odota kaksi sekuntia
    time.sleep(2)
```

Ohjelman suorittamisen aikana huomattiin REPL-komentokehotteesta, että luettu arvo muuttuu, kun potentiometriä käännetään.

3.1.7 D/A-muuntimen testaaminen

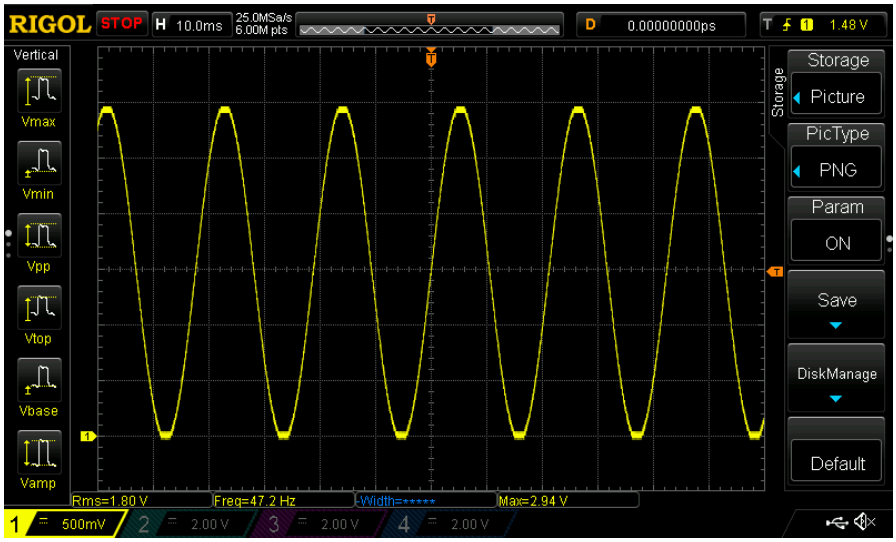
D/A-muuntimen testissä pyrittiin tuottamaan mahdollisimman korkeataajuista siniaaltoa, jotta voidaan vertailla eri ohjelmointiympäristöjen suoritusnopeutta. Tuhat siniaallon pistettä laskettiin taulukkomuuttujaan valmiiksi. Ohjelmassa vaihdettiin pinnin jännitettä while-silmukassa niin nopeasti kuin mahdollista (Kuva 15). Oskilloskoopilla mitattiin siniaallon taajuutta. Taajuuden perusteella voitiin laskea, kuinka monta kertaa sekunnissa jännitteen arvo vaihtui. Esimerkiksi aallon taajuuden ollessa yhden hertsin tarkoittaa se sitä, että jännitteen arvo päivittyi tuhat kertaa sekunnissa. MicroPythonilla päästiin 47,2 hertsin taajuuteen, joka tarkoittaa, että arvoa päivitettiin 47200 kertaa sekunnissa (Kuva 16). MicroPythonista löytyy myös oikosiirtoa hyödyntävä D/A-muuntimelle kirjoitukseen

tarkoitettu funktio. Oikosiirto (Direct Memory Access) on tekniikka, jolla voidaan siirtää dataa suoraan laitteiden välillä. Tällöin ei käytetä prosessorin resursseja, ja prosessorilla voidaan suorittaa muita tehtäviä tiedon välittämisen aikana (Gillis, n.d.). Tällä funktiolla päästiin huomattavasti suurempaan yli kymmenen kilohertsin taajuuteen (Kuva 17).

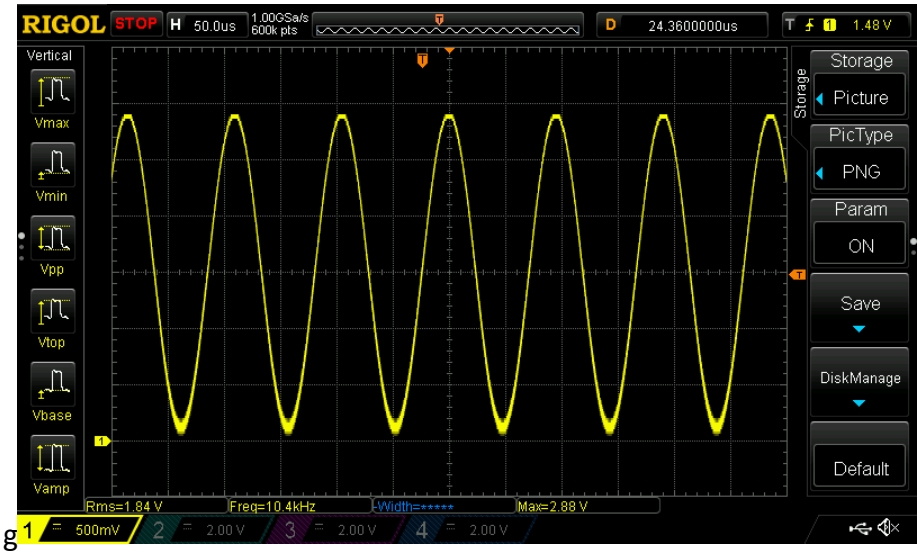
Kuva 15. Siniaallon muodostaminen MicroPythonilla.

```
# Tarvittavat tuonnit
import math
from pyb import DAC, delay
indeksi = 0
arvot = [0] * 1000
# DAC alustus
dac = DAC(2)
dac.init(bits = 8, buffering = True)

# Siniaallon pisteiden laskeminen taulukkoon
for i in range(len(arvot)):
    arvot[i] = 128 + int(127 * math.sin(2 * math.pi * i / len(arvot)))

# Funktio jännitteen muuttamiseen
def DAC_testi():
    global indeksi
    dac.write(arvot[indeksi])
    if indeksi >= (len(arvot) - 1):
        indeksi = 0
    else:
        indeksi += 1

# Ohjelman suoritus
while(1):
    DAC_testi()
```

Kuva 16. Oskilloskoopilla taajuuden mittaaminen.



Kuva 17. Siniaallon tuottaminen oikosiirtoa hyödyntäen.



g

3.1.8 Toimintojen ajoittaminen

Toimintojen ajoittamista testattiin suorittamalla eri toimintoja eri ajoituksella. Yhtä toimintoa suoritettiin jatkuvasti kymmenen sekuntia kehitysalustan käynnistyksen jälkeen, toista toimintoa suoritettiin kahden sekunnin välein ja kolmatta toimintoa kahden minuutin

välein. Jatkuvaksi toiminnoksi valittiin D/A-muuntimella sinikäyrän muodostaminen vastaavalla tavalla kuin D/A-muuntimen testaamisessa, kahden sekunnin välein vaihdettaisiin kehitysalustan sisäänrakennetun ledin tilaa ja kahden minuutin välein tulostettaisiin komentokehotteeseen tekstiä. Ajoittamiseen käytettiin MicroPythonin Timer-kirjastoa, joka mahdollisti toimintojen ajoittamisen helposti. Toimintojen ajoittamista testaava ohjelmakoodi on esitetty kuvassa 18.

Kuva 18. Ohjelmakoodi toimintojen ajoituksen testaamiseen MicroPythonilla.

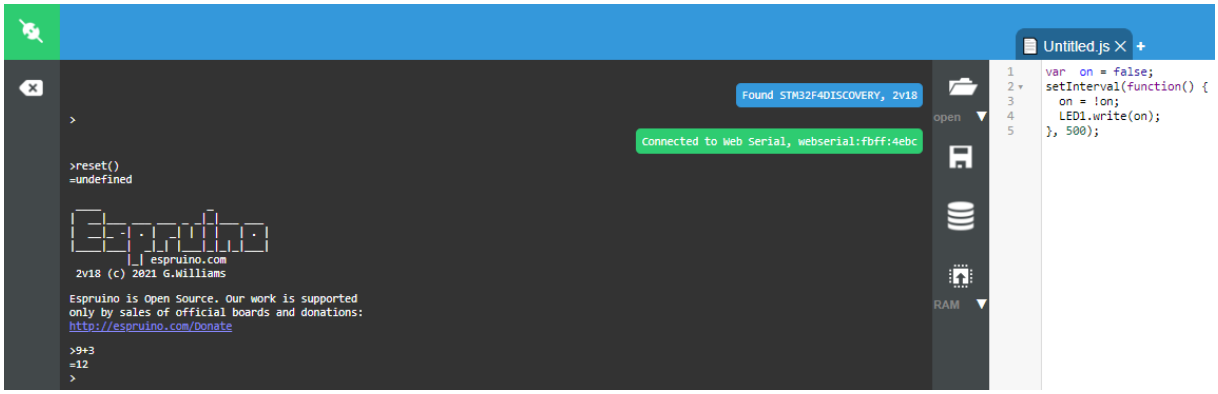
```
# Tarvittavat tuonnit
import math
from pyb import DAC, Timer, LED

# Muuttujien luonti ja DAC:n alustus
laskuri = 0
indeksi = 0
arvot = [0] * 100
dac = DAC(2)
dac.init(bits = 8, buffering = True)

# Ajastimien luominen ja alustus halutulle taajuudelle
ajastin1 = Timer(1, freq = 0.5)
ajastin2 = Timer(2, freq = 1/10)
ajastin3 = Timer(3, freq = 1000)
ajastin4 = Timer(4, freq = 1)

# Siniaallon pisteiden laskeminen
for i in range(len(arvot)):
    arvot[i] = 128 + int(127 * math.sin(2 * math.pi * i / len(arvot)))

# Funktio ledin tilan muuttamiselle
def led_vilkutus(timer):
    LED(1).toggle()

# Funtkio jolla aloitetaan DAC
def aloita_DAC(timer):
    ajastin3.callback(DAC)
    ajastin2.deinit()

# Funktio DAC:n arvon muuttamiselle
def DAC(timer):
    global indeksi
    dac.write(arvot[indeksi])
    if indeksi >= 99:
        indeksi = 0
    else:
        indeksi += 1

# Funktio tekstin tulostamiselle
def printtaa_hello(timer):
    global laskuri
    if laskuri == 120:
        print("Hello World!")
        laskuri = 0
    else:
        laskuri += 1

# Kutsuttavien funktioiden asetus
ajastin1.callback(led_vilkutus)
ajastin2.callback(aloita_DAC)
ajastin4.callback(printtaa_hello)
```

3.2 Espruinon testaaminen

3.2.1 Käyttöönotto

Espruinon käyttöönotto aloitettiin tutustumalla Espruinon verkkosivuihin. Verkkosivuilta ladattiin STM32F4DISCOVERY-kehitysalustalle sopiva Espruino versio binääritiedostona, joka ohjelmoitiin kehitysalustalle. Ohjelmointi tapahtui täysin vastaavasti kuin MicroPythonin ohjelmointi STM32CubeProgrammer-ohjelman avulla. Ohjelmoimisen jälkeen USB-johdot kytkettiin irti tietokoneesta ja takaisin. Tämän jälkeen asennuksen onnistuminen todettiin Espruino Web IDE:n avulla. Espruino Web IDE on suositeltu tapa kehitysalustan ohjelmointiin ja se tarjoaa myös helpon pääsyn REPL-komentokehotteeseen, joka toimii samalla tavoin kuin MicroPythonin REPL-komentokehote. Espruino Web IDE:llä yhdistämisen jälkeen huomataan komentokehotteessa Espruino teksti ja voidaan samalla testata komentokehotteen toimivuutta yksinkertaisella laskutoimituksella (Kuva 19). Kuvasta nähdään myös, että Espruino Web IDE:llä voidaan kirjoittaa ohjelmakoodia ja ohjelmoida kehitysalustaa.

Kuva 19. Espruino Web IDE:n näkymä.



3.2.2 Oman ohjelman ohjelmointi alustalle

Oman ohjelman siirtäminen kehitysalustalle on helppoa Espruino Web IDE:n ohjelmointipainiketta hyödyntäen. Painike näkyy edellä esitetyssä kuvakaappauksessa oikealla puolella kuvaa (Kuva 19). Painikkeesta pystytään valitsemaan ohjelmoidaanko flash-

vai ram-muistia. Napin painalluksen jälkeen kehitysalusta aloittaa ohjelman suorituksen ja tulosteet näkyvät komentokehotteessa. Esimerkkiohjelmana vilkutetaan LED-valoa viisi kertaa, kuten toimittiin MicroPythonia testattaessa (Kuva 20). Kytkennät olivat myös samat.

Kuva 20. LED-valon vilkutus Espruinolla.

```
// Led pinni
var led = A1;
// Sisääntulo pinni
var input = A5;
// Haluttu vilkutuskertojen määrä
var kerrat = 10;
// Aloitusindeksi
var indeksi = 0;
// Sisääntulon arvon muuttuja
var arvo;

function vilkutaLED() {
  if (indeksi % 2 === 0) {
    digitalWrite(led, true);
    arvo = digitalRead(input);
    print("Sisääntulon arvo:" + arvo);
  } else {
    digitalWrite(led, false);
    arvo = digitalRead(input);
    print("Sisääntulon arvo:" + arvo);
  }

  indeksi++;

  // Lopeta intervalli
  if (indeksi >= kerrat) {
    clearInterval(intervalID);
  }
}

// Intervallimuuttuja ledin sytyttämiselle
var intervalID = setInterval(vilkutaLED, 1000);

```

Pinnien nimeäminen poikkesi hieman Espruinossa verrattuna MicroPythoniin, mutta pinnien nimet voitiin tarkistaa Espruinon dokumentaatiosta. Sisääntulon arvo voitiin lukea komentokehotteesta (Kuva 21).

Kuva 21. Sisääntulon arvo.

3.2.3 Laskentanopeuden testaaminen

Suorituskykyä mitattiin samalla laskutoimituksella kuin MicroPythonia, eli Leibnizin kaavaa hyödyntäen laskettiin piin likiarvoa. Leibnizin kaava kirjoitettiin JavaScript ohjelmakoodiksi ja siirrettiin kehitysalustalle (Kuva 22). Laskennan tulokset tulostuivat komentokehotteeseen (Kuva 23).

Kuva 22. Piin likiarvon laskeminen Espruinolla.

```
1
2   // Funktio piin arvon laskemiseksi
3   function laske_pii(n)
4   {
5     print("Aloitetaan piin laskeminen...");
6     let pii = 0.0;
7     let osoittaja = 1;
8     // Muuttuja aloitushetken taltiointiin (sekunteina)
9     let aloitus = getTime();
10    for(let i = 0; i < n; i++){
11      if(i % 2 == 0){
12        pii += 4 / osoittaja;
13        osoittaja += 2;
14      }
15      else{
16      pii -= 4 / osoittaja;
17      osoittaja += 2;
18      }
19    }
20    // Muuttuja lopetushetken taltiointiin (sekunteina)
21    let lopetus = getTime();
22    print("Kulunut aika minuutteina: " + (lopetus-aloitus)/60);
23    print("Kulunut aika sekunteina: " + (lopetus-aloitus));
24    print("Kulunut aika millisekunteina: " + (lopetus-aloitus)*1000);
25    print("Piin arvo: " + pii);
26  }
27
28  // Aloitetaan laskenta
29  setTimeout(function(){laske_pii(1000000);}, 1000);
```

Kuva 23. Laskennan tulokset.

```
 ____                 _
|  __|___ ___ ___ _ _|_|___ ___
|  __|_ -| . |  _| | | |   | . |
|____|___|  _|_| |___|_|_|_|___|
        |_| espruino.com
 2v18 (c) 2021 G.Williams

Espruino is Open Source. Our work is supported
only by sales of official boards and donations:
http://espruino.com/Donate

>

Aloitetaan piin laskeminen...
Kulunut aika minuutteina: 6.05289740562
Kulunut aika sekunteina: 363.17384433746
Kulunut aika millisekunteina: 363173.84433746337
Piin arvo: 3.14159165358
```

Iteraatiokierroksia oli myös Espruinolla laskettaessa miljoona, kuten MicroPythonillakin laskettaessa. Laskenta suoritettiin viisi kertaa ja tulokset taulukoitiin.

3.2.4 Pulssinleveysmodulaation testaaminen

Pulssinleveysmodulaatiota lähdettiin testaamaan täysin samalla kytkennällä kuin MicroPython esimerkissä. Vastaava toiminta toteutettiin JavaScript ohjelmakoodilla (Kuva 24). Ohjelmalla siis LED-valo kirkastuu, kunnes saavuttaa täyden kirkkauden, jonka jälkeen se himmenee. Ohjelmaa suoritettaessa huomattiin sama käyttäytyminen kuin MicroPythonilla.

Kuva 24. Pulssinleveysmodulaatio Espruinolla.

```
let i = 0.1;
setInterval(function() {
  // Muutetaan pulssinleveyttä 0.1 - 1
  if(i < 1.0)
  {
    i = i + 0.1;
  }
  else
  {
    i = 0.1;
  }

  // Pinni A1 (PA1), pulssinleveys i, taajuus 1000 Hz
  analogWrite(A1, i, {freq : 1000});
}, 100);

```

3.2.5 Serial Peripheral Interface (SPI):n lukeminen

SPI-ominaisuutta testattiin Espruinolla täysin samalla järjestelyllä kuin MicroPythonilla. SPI-väylän lukemiseen tarvittava ohjelmakoodi kirjoitettiin JavaScriptillä Espruinolle, jonka jälkeen ohjelma siirrettiin kehitysalustalle (Kuva 25). Ohjelmakoodissa ei ole erikseen määritelty SPI-väylän nopeutta eikä moodia, mutta ne olisi voinut määritellä samalla metodilla, millä pinnien määritys tehtiin. Lämpötila tulostui komentokehotteeseen kahden sekunnin välein (Kuva 26).

Kuva 25. SPI-väylän lukeminen Espruinolla.

```
// Otetaan SPI1 väylä käyttöön. Pinni A5 - Serial Clock, A6 - Master In Slave Out,
// A7 - Master Out Slave in
SPI1.setup({sck:A5, miso:A6, mosi:A7});

// Pinni A1 - Chip Select
let cs = A1;

// Funktio lämpötilan lukemiseen
function lue_lampotila() {
  digitalWrite(cs, 0);
  // Luetaan SPI data
  let data = SPI1.send([0, 0]);
  digitalWrite(cs, 1);
  // Puretaan data
  let arvo = (data[0] << 8) | data[1];
  let lampotila = (arvo >> 3) * 0.25;

  print("Lämpötila: " + lampotila + "°C");
}

// Luetaan lämpötila kahden sekunnin välein
setInterval(lue_lampotila, 2000);
```

Kuva 26. Lämpötilan arvot terminaalista luettuna.

```
 _____                 _
|   __|___ ___ ___ _ _|_|___ ___
|__   | . |  _| | | | |   | . |
|_____|  _|_| |___|_|_|_|_|___|
      |_| espruino.com
 2v18 (c) 2021 G.Williams

Espruino is Open Source. Our work is supported
only by sales of official boards and donations:
http://espruino.com/Donate

>

Lämpötila: 26.5°C
Lämpötila: 26.25°C
Lämpötila: 26.5°C
```

3.2.6 A/D-muuntimen testaaminen

A/D-muunninta testattiin täysin samalla järjestelyllä kuin MicroPythonilla. Potentiometrin ulostulon arvon lukemaa tulostettiin komentokehotteeseen. A/D-muuntimen lukemiseen vaadittava ohjelma kirjoitettiin JavaScriptillä Espruinolle (Kuva 27). Ohjelman ajamisen jälkeen huomattiin arvojen muuttuvan komentokehotteessa, kun potentiometriä säädettiin.

Kuva 27. A/D-muuntimen käyttö Espruinolla.

```
setInterval(function() {
// Luetaan pinnistä A1 arvo väliltä 0.0 - 1.0
let arvo = analogRead(A1);
print(arvo);
}, 2000);
```

3.2.7 D/A-muuntimen testaaminen

D/A-muunninta pyrittiin testaamaan Espruinolla samalla tavoin kuin MicroPythonilla, mutta Espruino ei mahdollista pitkäaikaista while-silmukkaa ilman keskeytyksiä, joten Espruinolla testattaessa keskeytyksen arvoa pienennettiin, kunnes siniaallon taajuus ei enää kasvanut (Kuva 28). Siniaallon taajuudessa päästiin 1,2 hertsiin, eli 1200 näytettä sekunnissa (Kuva 29). Espruino tarjoaa myös aallon tekemiseen oman funktionsa, jolla päästiin 10 hertsin taajuuteen.

Kuva 28. Siniaallon muodostaminen Espruinolla.

```
// Muuttujien alustus
const arvot = [];
const arvojen_maara = 1000;
let indeksi = 0;

// Siniaallon pisteiden laskeminen taulukkoon
for(let i = 0; i <= arvojen_maara - 1; i++)
{
  arvot[i] = 0.5 + (0.5 * Math.sin(2 * Math.PI * i / arvojen_maara));
}

// Funktio jännitteen muuttamiseen
function DAC_testi()
{
  analogWrite(A5, arvot[indeksi]);
  if(indeksi >= arvojen_maara - 1)
  {
  indeksi = 0;
  }
  else
  {
   indeksi++;
  }
}

// Ohjelman suoritus silmukassa
setInterval(function(){DAC_testi();}, 0.1);
```

3.2.8 Toimintojen ajoitus

Espruinolla toimintojen ajoitus tehtiin käyttämällä sisäänrakennettuja funktiota setInterval ja setTimeout. setInterval funktio suorittaa silmukassa annettua funktiota halutun intervallin välein, kun taas setTimeout funktio suorittaa annetun funktion kerran annetun ajan jälkeen. Intervallin suorittaminen voidaan lopettaa clearInterval funktiolla.

Toimintojen ajoitusta testattiin samoilla toiminnoilla kuin MicroPythonilla. Ohjelmakoodi on esitetty seuraavassa kuvassa (Kuva 29).

Kuva 29. Toimintojen ajoituksen testaaminen Espruinolla.

```
const arvot = [];
const arvojen_maara = 100;
let indeksi = 0;
let on = false;

// Funktio signaalin muodostamiseksi
function DAC()
{
  analogWrite(A5, arvot[indeksi]);
  if(indeksi >= 100)
  {
  indeksi = 0;
  }
  else
  {
   indeksi++;
  }
}

//Funktio, joka aloittaa signaalin muodostuksen
function aloita_DAC()
{
  // Aloitetaan DAC 1000Hz taajudella
  setInterval(function(){DAC();}, 1);
  print("DAC aloitettu");
}

function printtaa_hello()
{
  print("Hello World!");
}

// Muutetaan kehitysalustan LED-valon tilaa
function led_vilkutus()
{
  on = !on;
  LED1.write(on);
}

// Luodaan siniaallon pisteet
for(let i = 0; i <= arvojen_maara; i++)
{
  arvot[i] = 0.5 + (0.5 * Math.sin(2 * Math.PI * i / arvojen_maara));
}
print("Pisteet laskettu muuttujaan");
print("Aloitetaan signaalin tuottaminen kymmenen sekunnin kuluttua...");

// Kutsuu kymmenen sekunnin kuluttua aloita_DAC-funktiota kerran
setTimeout(function(){aloita_DAC();}, 10000);
// Luodaan intervalli, joka kutsuu led_vilkutus-funktiota kahden sekunnin välein
setInterval(function(){led_vilkutus();}, 2000);
// Luodaan intervalli, joka kutsuu printtaa_hello-funktiota kahden minuutin välein
setInterval(function(){printtaa_hello();}, (60000*2));
```

3.3 Arduino-ohjelmoinnin testaaminen

Tässä kappaleessa käsitellään suunnitteluvaiheessa esitettyjä testattavia asioita Arduino-ohjelmointikieltä hyödyntäen. Tässä kappaleessa kielestä käytetään nimitystä C++, vaikkakin tarkalleen ottaen kieli ei ole täysin sama, mutta se pohjautuu siihen erittäin vahvasti.

3.3.1 Käyttöönotto

Käyttöönotto aloitettiin lataamalla ja asentamalla Arduinon verkkosivuilta Arduino IDE. Tämän jälkeen piti vielä asentaa tuki STM32-alustoille, koska Arduino IDE ei tue STM32-alustoja vakiona. Asennukseen löytyi selkeät ohjeet internetistä. Tuen asentamisen jälkeen kehitysalusta kytkettiin USB-johdolla ja Arduino IDE:n asetuksista valittiin STM32F4DISCOVERY-kortti sekä portti, johon kehitysalusta on yhdistetty.

3.3.2 Oman ohjelman ohjelmointi kehitysalustalle

Ohjelman kirjoittaminen ja siirtäminen alustalle onnistui suoraan Arduino IDE:stä. Upload-painiketta painamalla ohjelma siirretään kehitysalustalle. Siirrossa kesti kuitenkin jonkin aikaa, sillä Arduino IDE kääntää ohjelman konekielelle ennen kuin se siirretään alustalle. Esimerkkiohjelmassa vilkutettiin LED-valoa viisi kertaa ja luettiin LED-valon tilaa, kuten tehtiin muidenkin ympäristöjen esimerkkiohjelmassa. Kytkennät olivat myös samat. Esimerkkiohjelma kirjoitettiin C++-kielellä (Kuva 30). Tulostuksia voidaan tarkkailla millä tahansa sarjaporttiliikenteeseen kykenevällä terminaaliohjelmalla (Kuva 31).

Kuva 30. LED-valon vilkutus Arduinolla.

```
void setup() {
  // Avataan sarjaporttiyhteys nopeudella 9600
  Serial.begin(9600);
  int led = PA1;
  int input = PA5;
  int arvo = 0;
  // Pinnin asettaminen ulostuloksi
  pinMode(led, OUTPUT);
  // Pinnin asettaminen sisääntuloksi
  pinMode(input, INPUT);
  delay(1000);
  for(int i = 0; i < 10; i++)
  {
    if(i % 2 == 0)
    {
      digitalWrite(led, HIGH);
      arvo = digitalRead(input);
      Serial.println("Sisaantulon arvo: " + (String)arvo);
    }
    else
    {
      digitalWrite(led, LOW);
      arvo = digitalRead(input);
      Serial.println("Sisaantulon arvo: " + (String)arvo);
    }
    delay(1000);
  }
```

Kuva 31. Sisääntulon arvot.

```
Sisaantulon arvo: 1
Sisaantulon arvo: 0
Sisaantulon arvo: 1
Sisaantulon arvo: 0
Sisaantulon arvo: 1
Sisaantulon arvo: 0
Sisaantulon arvo: 1
Sisaantulon arvo: 0
Sisaantulon arvo: 1
Sisaantulon arvo: 0
```

3.3.3 Laskentanopeuden testaaminen

Laskentanopeutta testattiin samalla Leibnizin piin likiarvon kaavalla, jota käytettiin muidenkin ympäristöjen testaamiseen. Kaava kirjoitettiin C++-ohjelmakoodiksi, jonka jälkeen suoritettiin laskentatesti (Kuva 32).

Kuva 32. C++-koodi piin likiarvon laskemiseksi.

```
void setup() {
  delay(1000);
  // Sarjaporttiyhteys nopeudella 9600
  Serial.begin(9600);
  Serial.println("Aloitetaan piin laskeminen...");
  // Aloitusaika
  int aloitus = millis();
  float pii = laske_pii(1000000);
  // Lopetusaika
  int lopetus = millis();
  // Kulunut aika
  int erotus = lopetus - aloitus;
  Serial.print("Piin arvo: ");
  Serial.println(pii, 6);
  Serial.print("Kulunut aika millisekunteina: ");
  Serial.println(erotus);
}
void loop(){}
// Funktio piin likiarvon laskemiseksi
float laske_pii(int n)
{
  float pii = 0.0;
  float osoittaja = 1.0;
  for(int i = 0; i < n; i++)
  {
    if(i % 2 == 0)
    {
      pii += 4 / osoittaja;
      osoittaja += 2;
    }
    else
    {
      pii -= 4 / osoittaja;
      osoittaja += 2;
    }
  }
  return pii;
}
```

Terminaaliohjelmasta voitiin huomata, että piin arvossa viisi ensimmäistä desimaalia pilkun jälkeen oli oikein ja piin laskemiseen kului 139 millisekuntia. Laskennassa käytettiin kääntäjän oletuksena käyttämää optimointivalintaa. Laskenta toistettiin viisi kertaa ja tulokset taulukoitiin.

3.3.4 Pulssinleveysmodulaation testaaminen

Pulssinleveysmodulaatiota testattiin vastaavilla kytkennöillä, kuin muitakin ohjelmointiympäristöjä. Ohjelmakoodi kirjoitettiin C++-kielelle ja siirrettiin kehitysalustalle (Kuva 33). Arduino-ohjelmointikielen dokumentaatiota tutkiessa huomattiin, ettei Arduino mahdollista pulssinleveyden taajuuden muuttamista oletuksena, mutta STM32-tuen asentamisen myötä sekin olisi mahdollista.

Kuva 33. Pulssinleveysmodulaatio Arduinolla.

```
// Pinnin valinta
int led = PA1;
void setup() {
  // Pinnin määritys ulostuloksi
  pinMode(led, OUTPUT);
}
int i = 1;
// Muutetaan pulssinleveyden arvoa (1-255)
void loop() {
  if(i < 255)
  {
    i++;
  }
  else
  {
    i = 1;
  }
  analogWrite(led, i);
  delay(10);
}
```

3.3.5 Serial Peripheral Interface (SPI):n lukeminen

SPI:n testaaminen suoritettiin vastaavilla kytkennöillä kuin muitakin ohjelmointiympäristöjä testattaessa. Ohjelmakoodi lämpötilan lukemiseksi kirjoitettiin C++-ohjelmointikielelle ja se siirrettiin kehitysalustalle (Kuva 34). Ohjelman suorittamisen jälkeen voitiin tarkastella lämpötilaa terminaaliohjelmasta onnistuneesti.

Kuva 34. SPI väylän lukeminen Arduinolla.

```
#include <SPI.h>
// Määritetään CS (Chip Select) pinni
int CS = PA1;
float lampotila = 0;
void setup() {
  // Avataan sarjaporttiyhteys nopeudella 9600
  Serial.begin(9600);
  pinMode(CS, OUTPUT);
  digitalWrite(CS, HIGH);
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
}
void loop() {
  lampotila = lue_lampotila();
  // Tulostetaan lämpötilan arvo
  Serial.println(lampotila);
  // Odotetaan kaksi sekuntia
  delay(2000);
}

// Lämpötilan lukeminen
float lue_lampotila()
{
  uint16_t data = 0;
  digitalWrite(CS, LOW);
  delay(10);
  // Luetaan data SPI väylästä
  data = SPI.transfer16(0x00);
  digitalWrite(CS, HIGH);
  Serial.println(data);
  // Muunnetaan luettu tieto lämpötilaksi
  float arvo = (data >> 3) * 0.25;
  return arvo;
}
```

3.3.6 A/D-muuntimen testaaminen

A/D-muunninta testattiin Arduinolla vastaavasti kuin muita ohjelmointiympäristöjä. A/D-muuntimen arvon lukemiseen vaadittava ohjelma kirjoitettiin C++-ohjelmointikielelle ja se suoritettiin kehitysalustalla (Kuva 35).

Kuva 35. A/D-muuntimen lukeminen Arduinolla.

```
// Alustetaan muuttujat
int adc = PA1;
int arvo = 0;
void setup() {
  // Avataan sarjaporttiyhteys
  Serial.begin(9600);
}

void loop() {
  // Luetaan arvo ja tulostetaan se
  arvo = analogRead(adc);
  Serial.println(arvo);
}
```

3.3.7 D/A-muuntimen testaaminen

D/A-muunninta testattiin myös Arduinolla vastaavalla tavalla kuin muita ohjelmointiympäristöjä (Kuva 36). Arduinolla päästiin 270 hertsin taajuuteen ilman kääntäjän optimointia, ja kun käännöstä optimoitiin mahdollisimman nopeaksi, päästiin 417 hertsin taajuuteen. Eli päivitystaajuus Arduinossa oli 270000–417000 riippuen kääntäjän asetuksista.

Kuva 36. Siniaallon muodostaminen Arduinolla.

```
// Muuttujien alustus
int arvojen_maara = 1000;
float arvot[1000] = {0};
int indeksi = 0;

void setup() {
  // Siniaallon pisteiden laskeminen taulukkoon
  for(int i = 0; i <= arvojen_maara - 1; i++)
  {
    arvot[i] = 128 + (127 * sin(2 * M_PI * i / arvojen_maara));
  }
}

// Ohjelman suorittaminen
void loop() {
  DAC_testi();
}

// Funktio jännitteen muuttamiseen
void DAC_testi()
{
  analogWrite(PA5, arvot[indeksi]);
  if(indeksi >= arvojen_maara - 1)
  {
  indeksi = 0;
  }
  else
  {
   indeksi++;
  }
}
```

3.3.8 Toimintojen ajoittaminen

Oletuksena Arduino-ohjelmointiympäristö ei tarjoa kovin hyviä työkaluja usean toiminnon ajoittamiseen. Yleinen tapa on laskea kulunutta aikaa millisekunteina ja if-käskyllä toteuttaa haluttuja toimintoja tiettyinä ajan hetkinä. Arduinolle on kuitenkin olemassa Arduino timer-kirjasto, joka helpottaa toimintojen ajoitusta. Alla olevassa kuvassa on esitetty toimintojen ajoitus Arduino timer-kirjastoa käyttäen (Kuva 37). Ohjelman toiminta oli vastaava kuin muitakin ohjelmointiympäristöjä testattaessa.

Kuva 37. Toimintojen ajoitus Arduinolla.

```
#include <arduino-timer.h>
// Luodaan ajastin
auto timer = timer_create_default();

// Muuttujat siniaaltoa varten
float arvot[100] = {0};
int arvojen_maara = 100;
int indeksi = 0;

// Funktio signaalin muodostamiseksi
bool DAC_siniaalto(void *)
{
  analogWrite(PA5, arvot[indeksi]);
  if(indeksi >= arvojen_maara)
  {
  indeksi = 0;
  }
  else
  {
   indeksi++;
  }
  return true;
}

// Funktio, jolla aloitetaan signaalin muodostus
bool aloita_DAC(void *)
{
  // Luodaan siniaallon pisteet
  for(int i = 0; i <= arvojen_maara; i++)
  {
    arvot[i] = 128 + (127 * sin(2 * M_PI * i / arvojen_maara));
  }
  // Aloitetaan DAC 1000Hz taajudella
  timer.every(1, DAC_siniaalto);
  Serial.println("DAC aloitettu");
  return false;
}

// Muutetaan kehitysalustan LED-valon tilaa
bool led_vilkutus(void *)
{
  digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
  return true;
}

// Tulostetaan "Hello World"
bool printtaa_hello(void *)
{
  Serial.println("Hello World!");
  return true;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
  Serial.println("Aloitetaan signaalin tuottaminen kymmenen sekunnin kuluttua...");
  // Kutsuu kymmenen sekunnin kuluttua aloita_DAC-funktiota kerran
  timer.in(10000, aloita_DAC);
  // Kutsutaan led_vilkutus-funktiota kahden sekunnin välein
  timer.every(2000, led_vilkutus);
  // Kutsutaan printtaa_hello-funktiota kahden minuutin välein
  timer.every(120000, printtaa_hello);
}

void loop() {
  // Aloitetaan ajastin
  timer.tick();
}
```

4 Arviointi

4.1 Ohjelmointiympäristöjen käyttöönotto

Kaikki kolme eri ympäristöä olivat varsin yksinkertaisia ottaa käyttöön. Espruino ja MicroPython tarvitsivat STM32CubeProgrammer-ohjelman ja Arduino Arduino IDE-ohjelman ympäristön käyttöönottamiseksi, joten kaikkien eri ympäristöjen käyttöönotto vaatii jonkinlaisen ohjelman latauksen. Asennukset olivat melko nopeita ja kevyitä.

Espruino ja MicroPython olivat identtisiä käyttöönotoltaan. Molemmissa tapauksissa verkosta piti ladata kehitysalustalle sopiva binääritiedosto, joka siirrettiin kehitysalustalle. Tämän jälkeen ympäristöt olivat käyttövalmiita. Arduinon kohdalla piti asentaa Arduino IDE-ohjelman kautta tuki STM32-mikrokontrollereille, mutta asennukseen löytyi verkosta hyvät ohjeet ja asennus oli helppo toteuttaa. Asennuksen jälkeen kehitysalustan ohjelmointi onnistui helposti, kuten minkä tahansa muunkin virallisen Arduino-kehitysalustan.

4.2 Laskentanopeuden arviointi

Kaikkien ohjelmointiympäristöjen laskentanopeutta testattiin laskemalla piin likiarvoa Leibnizin-kaavalla. Leibnizin kaavassa saadaan sitä tarkempi piin likiarvo, mitä pidemmälle laskentaa jatketaan. Kaikilla ohjelmointiympäristöillä suoritettiin miljoona kierrosta ja ohjelman rakenne pyrittiin pitämään niin samanlaisena kuin mahdollista eri ohjelmointiympäristöjen välillä, eli rakennetta ei optimoitu jokaiselle ohjelmointikielelle erikseen. Taulukoiduista mittaustuloksista laskettiin keskiarvot, jotka ovat esitetty taulukossa kaksi. Huomiona, että vain Espruinolla tehtyjen laskentojen suoritusajat vaihtelivat hieman.

Taulukko 2. Piin likiarvon laskemiseen kuluneet ajat.

Ohjelmointiympäristö	Kulunut aika (ms)
MicroPython	14538
Espruino	363174
Arduino	139

Taulukosta huomataan, että Arduino C++-kielellä oli selvästi nopein piin likiarvon laskemisessa. Tämä oli odotettavaa, sillä C++ on käännettävä kieli. Espruino ja MicroPython käyttävät tulkkia ohjelman suorittamiseen, mikä on hitaampaa. Tulkattavista vaihtoehdoista MicroPython oli selvästi Espruinoa nopeampi. Espruinolla kului aikaa melkein 25 kertaa kauemmin kuin MicroPythonilla. Kuitenkin MicroPython oli selvästi Arduinoa hitaampi, sillä MicroPythonilla kului laskennassa melkein 105 kertaa kauemmin kuin Arduinolla.

Espruinolla laskenta ei sujunut täysin ongelmitta. Espruinolla pitkään kestävää laskentaa ei voi tehdä ilman, että funktio asetetaan sisäänrakennettuun setTimeout-funktioon. Tämä aiheutti aluksi ongelmia, kunnes Espruinon foorumeilta selvisi, että mikäli interaktiivinen komentokehote ei saa vastausta tietyn ajan sisällä se kaatuu. Mikäli pitkään kestävä laskenta on setTimeout-funktion sisällä, ohjelma toimii kaatumatta. Muilla ohjelmointiympäristöillä ei ollut vastaavia ongelmia. Tämän laskentatehtävän perusteella Espruinoa ei voi suositella laskentaa vaativaan sovellukseen ja toisaalta MicroPythoniltakin kului aikaa huomattavasti enemmän kuin Arduinolta. Toisin sanoen suorituskykykriittisissä sovelluksissa käännettävä kieli on ylivoimainen mikä ei toisaalta tullut yllätyksenä.

4.3 Kehitysalustan ominaisuuksien käyttäminen

STM32F4DISCOVERY-kehitysalustan ominaisuuksista testattiin GPIO:ta, A/D-muunninta, D/A-muunninta, pulssinleveysmodulaatiota ja SPI-väylän lukemista. Kehitysalustan ominaisuuksien lisäksi testattiin eri toimintojen ajoittamista. Testattavat ominaisuudet olivat mikrokontrollereiden perusominaisuuksia, joten kaikista ohjelmointiympäristöistä löytyi ominaisuudet niiden hallintaan.

Pulssinleveysmodulaatiota pystyi vaivattomasti tuottamaan kaikilla ohjelmointiympäristöillä. Kaikilla ohjelmointiympäristöillä pystyi vaikuttamaan taajuuteen sekä pulssinleveyteen. Toisin kuin tavallisessa Arduinossa, on STM32-mikrokontrolleritukeen lisätty ominaisuus pulssin taajuuden muuttamiseen. SPI-väylän ja A/D-muuntimen käyttö oli lähes identtinen kaikilla ohjelmointiympäristöillä.

Ainoastaan MicroPythonissa on oma luokka D/A-muuntimen käyttämiseen. Muissa testattavissa ympäristöissä käytettiin analogWrite-funktiota D/A-muuntimen hyödyntämiseen, joka on siis sama funktio kuin pulssinleveysmodulaatiota käytettäessä. Espruinossa ja Arduinossa, jos pinni on kytketty analogiamuuntimeen, käytetään D/A-muunninta. Tässä mielessä MicroPythonin implementaatio oli selkeämpi. MicroPython ja Espruino tarjosivat aallon muodostamiseen omat funktiot, joista varsinkin MicroPythonin oikosiirtoa hyödyntänyt implementaatio pystyi selkeästi korkeimpaan näytteenottotaajuuteen (Taulukko 3). Taulukosta nähdään, että käännettävää kieltä hyödyntävä Arduino oli selvästi nopein, jos ei huomioida MicroPythonin oikosiirtoa hyödyntävää toteutusta. Oikosiirto on huomattavasti nopeampaa, koska tällöin siirrettävää tietoa ei kuljeteta prosessorin kautta. Arduino-ohjelmointiympäristö ei tarjoa vakiona funktiota sen hyödyntämiseen.

Taulukko 3. Näytteenottotaajuudet.

Ohjelmointiympäristö	Näytteenottotaajuus (1/s)
MicroPython	47200
MicroPython (oikosiirto)	10200000
Espruino	1200
Espruino (waveform-luokka)	10000
Arduino	270000-417000

Toimintojen ajoittaminen toimi jokaisessa ympäristössä hieman eri tavoin. Arduinossa setup-osio suoritetaan vain kerran, minkä jälkeen siirrytään loop-osioon, jota suoritetaan loputtomasti. Arduinossa tavallisesti käytetään millisekuntien laskemista ja kuluneiden millisekuntien vertaamista vertailuarvoon, kun halutaan suorittaa jokin toiminto. Tämä kuitenkin monimutkaistaa ohjelmakoodia, eikä siten ole kovinkaan käytännöllinen ratkaisu, joten Arduinon yhteydessä asennettiin kirjasto, joka mahdollistaa toimintojen ajoittamisen. Arduino ei myöskään oletuksena tue säikeiden luomista, mutta siihen on myös olemassa lisäosana asennettava kirjasto. Arduino ei siis oletuksena tarjoa kovin hyviä työkaluja toimintojen ajoittamiseen.

Espruinon toimintaperiaate eroaa muista siinä, että siinä oikeastaan kaikki toiminnot asetetaan setInterval- ja setTimeout-funktioiden sisään. Näillä samoilla funktiolla myös

toimintojen tarkka ajoittaminen on helppoa. Intervalleja voidaan myös lopettaa clearInterval-funktiolla. Espruinolla toimintojen ajoittaminen käy pienen totuttelun jälkeen helposti.

MicroPython tarjoaa vakiona ajastimien käytön. Ajastimille voidaan asettaa haluttu taajuus tai sitten käyttäjä voi skaalata kellotaajuuden oikeaksi ja asettaa halutun jakson. Yhdelle ajastimelle voi asettaa vain yhden funktion kutsuttavaksi, mikä tuntuu ajastimien tuhlaukselta.

Mikrokontrollereiden ohjelmointityöhön on olemassa myös ammattimaisempia ohjelmointiympäristöjä tässä opinnäytetyössä esiteltyihin verrattuna. Näiden opettelu on kuitenkin huomattavasti enemmän aikaa vievää. Tässä opinnäytetyössä käytetyn STMicroelectronics:n kehitysalustan ohjelmoimiseen on olemassa valmistajan tarjoama ohjelmointiympäristö STM32CubeIDE, jossa kehittäjä voi valita käyttääkö C- vai C++-ohjelmointikieltä (STMicroelecronics, n.d.b). Mikrokontrollerivalmistajien omat ohjelmointiympäristöt keskittyvät vain valmistajien omien tuotteiden ohjelmointiin eivätkä ole suoraan tarkoitettu muiden valmistajien tuotteiden ohjelmoimiseen.

5 Pohdinta

Tässä opinnäytetyössä pystyttiin tutkimaan ainoastaan murto-osa näiden ohjelmointiympäristöjen eroavaisuuksista ja ominaisuuksista. MicroPythonista ja Espruinosta, jotka edustivat tässä opinnäytetyössä vaihtoehtoisia ohjelmointikieliä, voisi molemmista tehdä oman opinnäytetyön ja perehtyä niihin syvemmin.

Erityisesti MicroPython oli positiivinen yllätys. Se olisi todella hyvä vaihtoehto mikrokontrollereiden ohjelmoinnin opettelussa. Python on laajasti opetettu kieli opiskelijoille, joten kynnys siirtyä MicroPythoniin olisi pieni. MicroPythonin REPL-komentokehotteella pystyy ohjaamaan kehitysalustaa nopeasti ja näkemään komentojen vaikutuksen välittömästi. Toisaalta C-ohjelmointikieli on edelleen niin suosittu sulatetuissa järjestelmissä, että se on käytännössä pakko opetella, jos aikoo työskennellä mikrokontrollereiden parissa. MicroPythonin yleisimmät käyttökohteet onkin useissa

lähteissä merkitty opetuskäytöksi sekä prototyyppien kehittämiseksi, mutta sitä voisi hyvin käyttää myös tuotteissa, joissa koodin suorituskyky ei vaadi käännetyn kielen tehokkuutta. MicroPythonin voi laajentaa käyttämään myös C-ohjelmointikielen funktioita, jolloin voidaan käyttää yleistason logiikan kirjoittamiseen selkeää ja helppolukuista MicroPythonia ja suorituskykykriittisten osioiden kirjoittamiseen C-ohjelmointikieltä.

Espruino tarjosi MicroPythonin tavoin REPL-komentokehotteen, joka auttaa tutustumaan ohjelmointiympäristöön, sillä komentojen vaikutuksen voi nähdä välittömästi. Molemmissa myös koodin muokkaus ja kokeilu on nopeaa, koska koodia ei tarvitse kääntää. Espruinoa voi myös ohjelmoida Espruinon Blockly-editorilla, jolloin ei tarvitse välttämättä osata kirjoittaa ohjelmakoodia vaan käyttää visuaalista Blockly-ohjelmointia. Molemmille kehitysympäristöille on olemassa virallisia kehitysalustoja ja Espruinolla jopa älykello, johon voi itse helposti lisätä ominaisuuksia. Kuitenkin huomattavasti laajempi tuki eri alustoille on MicroPythonilla. Molempia alustoja voi hyvin käyttää erilaiseen asioiden testailuun, opetteluun ja prototyyppien valmistukseen. Varsinaiseen tuotantoon on kuitenkin harkittava tarkkaan, että onko nämä vaihtoehtoiset ohjelmointiympäristöt soveltuvia.

Jatkokehityksenä voisi tehdä vaihtoehtoisilla ohjelmointikielillä jonkinlaisen laitteen, joka hyödyntäisi tästä opinnäytetyöstä pois jäänyttä verkkoliikennettä. MicroPython ja Espruino tarjoavat molemmat valmiit luokat verkkoliikenteen ja esimerkiksi MQTT-protokollan hyödyntämiseen. Molemmista löytyy myös luokat bluetoothin hyödyntämiseen.

Lähteet

Analog Devices. (n.d.). *MAX6675* [kuva]. *https://www.analog.com/en/products/max6675.html*

Dickson, B. (21.5.2022). *A Brief History of JavaScript. https://dev.to/dboatengx/history-of-javascript-how-it-all-began-92a*

Evanczuk, S. (24.11.2019). *Is it time to retire C?* [kuva]. https://www.embedded.com/2019-embedded-markets-study-reflects-emerging-technologies-continued-c-c-dominance/

Gillis, A. (n.d.). *Direct Memory Access (DMA)*. https://www.techtarget.com/whatis/definition/Direct-Memory-Access-DMA

Kirvan, P. (n.d.). *analog-to-digital conversion (ADC*). https://www.techtarget.com/whatis/definition/analog-to-digital-conversion-ADC

Kumari, B. (n.d.). *History of C Language. https://www.scaler.com/topics/c/history-of-c-language/*

Microchip. (n.d.). *Pulse-Width Modulation (PWM)* [kuva]. https://onlinedocs.microchip.com/pr/GUID-2B9A050B-5795-4BBF-BA25-BB1B1C2C6D10-en-US-3/index.html?GUID-CAC33046-81F0-4D4B-B41F-91DE2CDFB91C

MicroPython. (n.d.). *MicroPython.* https://micropython.org/

Nerdy Electorincs. (n.d.). *Introduction to GPIO*. https://nerdyelectronics.com/introduction-to-gpio/

Paramanick, S. (n.d.). *History of Python.* https://www.geeksforgeeks.org/history-of-python/

Python. (n.d.). *Application for Python. https://www.python.org/about/apps/*

Sparkfun. (n.d.). *Serial Peripheral Interface (SPI).* https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Statista. (2023). *Most used programming languages among developers worldwide as of 2023* [kuva]. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

STMicroelectronics. (n.d.a). *STM32F4DISCOVERY* [kuva]. https://www.st.com/en/evaluation-tools/stm32f4discovery.html

STMicroelectronics. (n.d.b). *Integrated Development Environment for STM32*. https://www.st.com/en/development-tools/stm32cubeide.html

University Of Plymouth. (n.d.). *Digital to Analogue Converter (DAC) – (Glossary Entry).* https://blogs.plymouth.ac.uk/embedded-systems/glossary-2/digital-to-analogue-converter-dac-glossary-entry/

W3Schools. (n.d.). *Python Introduction.* *https://www.w3schools.com/python/python_intro.asp*